ECCHacks:
a gentle introduction
to elliptic-curve cryptography

Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Tanja Lange
Technische Universiteit Eindhoven

`ecchacks.cr.yp.to`

Cryptography

Public-key signatures:
e.g., RSA, DSA, ECDSA.
Some uses: signed OS updates,
SSL certificates, e-passports.

Public-key encryption:
e.g., RSA, DH, ECDH.
Some uses: SSL key exchange,
locked iPhone mail download.

Secret-key encryption:
e.g., AES, Salsa20.
Some uses: disk encryption,
bulk SSL encryption.

:

troduction

curve cryptography

Bernstein

of Illinois at Chicago &

Universiteit Eindhoven

ge

Universiteit Eindhoven

.cr.yp.to

Cryptography

Public-key signatures:
e.g., RSA, DSA, ECDSA.
Some uses: signed OS updates,
SSL certificates, e-passports.

Public-key encryption:
e.g., RSA, DH, ECDH.
Some uses: SSL key exchange,
locked iPhone mail download.

Secret-key encryption:
e.g., AES, Salsa20.
Some uses: disk encryption,
bulk SSL encryption.

Why ECC?

"Index calc

to break or

Long histor

including n

1975, CFR

1977, linea

1982, quad

1990, num

1994, func

2006, medi

2013, x^q –

(FFS is not

graphy

Chicago &
Eindhoven

Eindhoven

Cryptography

Public-key signatures:
e.g., RSA, DSA, ECDSA.
Some uses: signed OS updates,
SSL certificates, e-passports.

Public-key encryption:
e.g., RSA, DH, ECDH.
Some uses: SSL key exchange,
locked iPhone mail download.

Secret-key encryption:
e.g., AES, Salsa20.
Some uses: disk encryption,
bulk SSL encryption.

Why ECC?

"Index calculus": fastes
to break original DH ar

Long history,
including many major i
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (
1990, number-field siev
1994, function-field siev
2006, medium-prime FF
2013, $x^q - x$ FFS "cry

(FFS is not relevant to

Cryptography

Public-key signatures:
e.g., RSA, DSA, ECDSA.
Some uses: signed OS updates,
SSL certificates, e-passports.

Public-key encryption:
e.g., RSA, DH, ECDH.
Some uses: SSL key exchange,
locked iPhone mail download.

Secret-key encryption:
e.g., AES, Salsa20.
Some uses: disk encryption,
bulk SSL encryption.

Why ECC?

"Index calculus": fastest method we
to break original DH and RSA.

Long history,
including many major improvements
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS "cryptopocalypse

(FFS is not relevant to RSA.)

Cryptography

Public-key signatures:
e.g., RSA, DSA, ECDSA.
Some uses: signed OS updates,
SSL certificates, e-passports.

Public-key encryption:
e.g., RSA, DH, ECDH.
Some uses: SSL key exchange,
locked iPhone mail download.

Secret-key encryption:
e.g., AES, Salsa20.
Some uses: disk encryption,
bulk SSL encryption.

Why ECC?

"Index calculus": fastest method we know
to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS "cryptopocalypse".

(FFS is not relevant to RSA.)

phy

signatures:
DSA, ECDSA.
: signed OS updates,
cates, e-passports.

encryption:
DH, ECDH.
: SSL key exchange,
one mail download.

encryption:
Salsa20.
: disk encryption,
ncryption.

Why ECC?

“Index calculus”: fastest method we know
to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS “cryptopocalypse”.

(FFS is not relevant to RSA.)

Also many
$\approx$ 100 scie

Approxima
for breakin
CFRAC: 2^1
LS: 2^1
QS: 2^1
NFS: 2

A.
updates,
ports.

change,
wnload.

tion,

Why ECC?

"Index calculus": fastest method we know to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS "cryptopocalypse".

(FFS is not relevant to RSA.)

Also many smaller impr
$\approx$ 100 scientific papers.

Approximate costs of th
for breaking RSA-1024,
CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

Why ECC?

"Index calculus": fastest method we know to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS "cryptopocalypse".

(FFS is not relevant to RSA.)

Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms for breaking RSA-1024, RSA-2048:
CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

Why ECC?

"Index calculus": fastest method we know to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS "cryptopocalypse".

(FFS is not relevant to RSA.)

Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms for breaking RSA-1024, RSA-2048:

CFRAC:	2^{120}, 2^{170}.
LS:	2^{110}, 2^{160}.
QS:	2^{100}, 2^{150}.
NFS:	2^{80}, 2^{112}.

Why ECC?

"Index calculus": fastest method we know to break original DH and RSA.

Long history,
including many major improvements:
1975, CFRAC;
1977, linear sieve (LS);
1982, quadratic sieve (QS);
1990, number-field sieve (NFS);
1994, function-field sieve (FFS);
2006, medium-prime FFS/NFS;
2013, $x^q - x$ FFS "cryptopocalypse".

(FFS is not relevant to RSA.)

Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms for breaking RSA-1024, RSA-2048:
CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

1985 Miller
"Use of elliptic curves in cryptography":
"It is extremely unlikely that an 'index calculus' attack on the elliptic curve method will ever be able to work."

?

culus": fastest method we know
riginal DH and RSA.

ry,
many major improvements:
AC;
r sieve (LS);
lratic sieve (QS);
ber-field sieve (NFS);
tion-field sieve (FFS);
ium-prime FFS/NFS;
x FFS "cryptopocalypse".

t relevant to RSA.)

Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms
for breaking RSA-1024, RSA-2048:

CFRAC:	2^{120}, 2^{170}.
LS:	2^{110}, 2^{160}.
QS:	2^{100}, 2^{150}.
NFS:	2^{80}, 2^{112}.

1985 Miller
"Use of elliptic curves in cryptography":
"It is extremely unlikely that an
'index calculus' attack on the elliptic
curve method will ever be able to work."

The clock

This is the

Warning:
This is *not*
"Elliptic cu

st method we know
nd RSA.

mprovements:

QS);
e (NFS);
ve (FFS);
FS/NFS;
ptopocalypse".

RSA.)

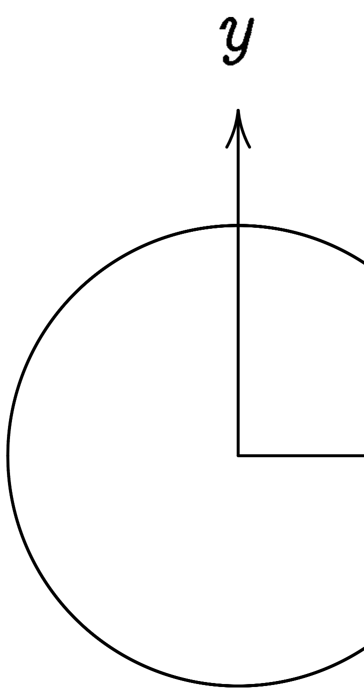
Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms
for breaking RSA-1024, RSA-2048:

CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

1985 Miller
"Use of elliptic curves in cryptography":
"It is extremely unlikely that an
'index calculus' attack on the elliptic
curve method will ever be able to work."

The clock



This is the curve $x^2 +$

Warning:
This is *not* an elliptic c
"Elliptic curve" $\neq$ "ellip

e know

s:

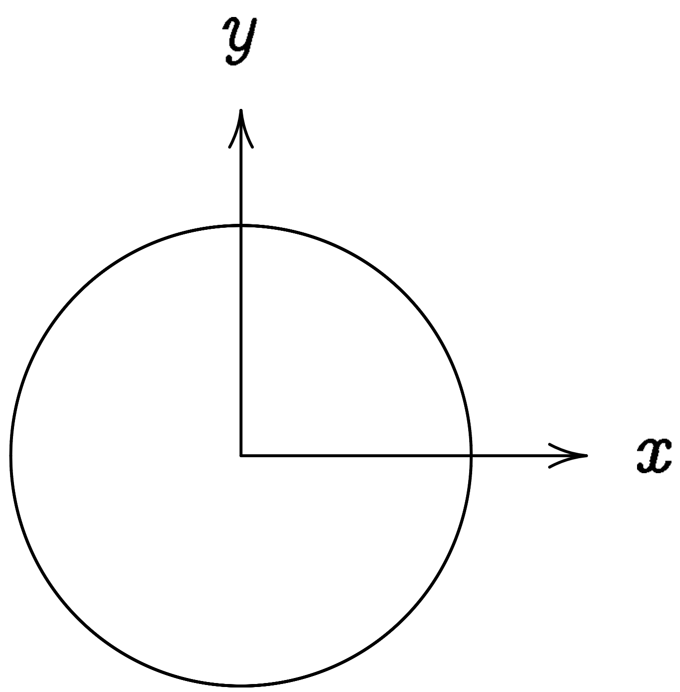
”.

Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms
for breaking RSA-1024, RSA-2048:
CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

1985 Miller
“Use of elliptic curves in cryptography”:
“It is extremely unlikely that an
‘index calculus’ attack on the elliptic
curve method will ever be able to work.”

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:
This is *not* an elliptic curve.
“Elliptic curve” $\neq$ “ellipse.”

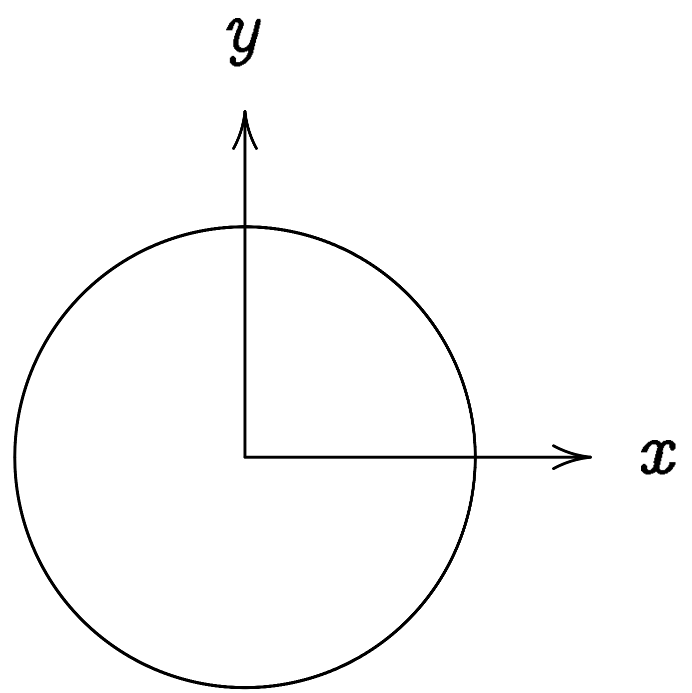
Also many smaller improvements:
$\approx$ 100 scientific papers.

Approximate costs of these algorithms
for breaking RSA-1024, RSA-2048:

CFRAC: 2^{120}, 2^{170}.
LS: 2^{110}, 2^{160}.
QS: 2^{100}, 2^{150}.
NFS: 2^{80}, 2^{112}.

1985 Miller
"Use of elliptic curves in cryptography":
"It is extremely unlikely that an
'index calculus' attack on the elliptic
curve method will ever be able to work."

The clock



This is the curve $x^2 + y^2 = 1$.

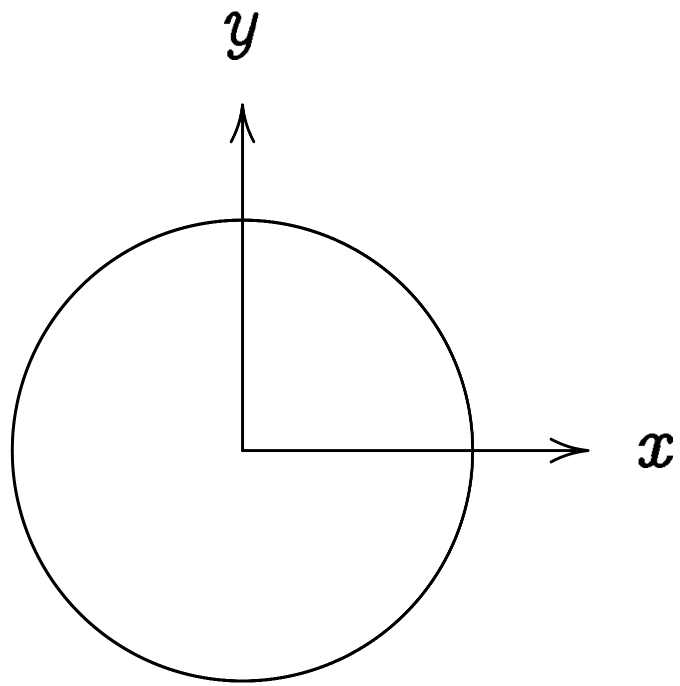
Warning:
This is *not* an elliptic curve.
"Elliptic curve" $\neq$ "ellipse."

smaller improvements:
ntific papers.

te costs of these algorithms
g RSA-1024, RSA-2048:
^{120}, 2^{170}.
^{110}, 2^{160}.
^{100}, 2^{150}.
^{80}, 2^{112}.

r
iptic curves in cryptography":
mely unlikely that an
ulus' attack on the elliptic
hod will ever be able to work."

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:
This is *not* an elliptic curve.
"Elliptic curve" $\neq$ "ellipse."

Examples c

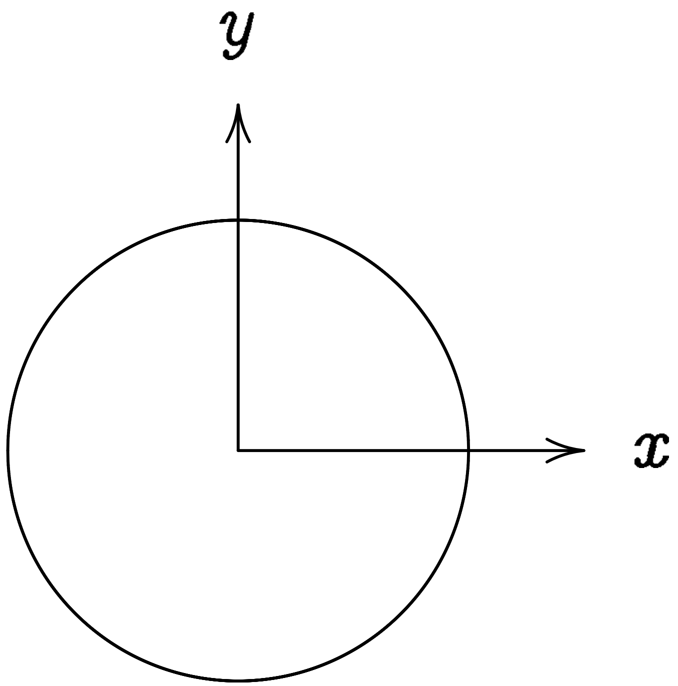
rovements:

.

hese algorithms
RSA-2048:

n cryptography":
that an
on the elliptic
be able to work."

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:
This is *not* an elliptic curve.
"Elliptic curve" $\neq$ "ellipse."

Examples of points on t

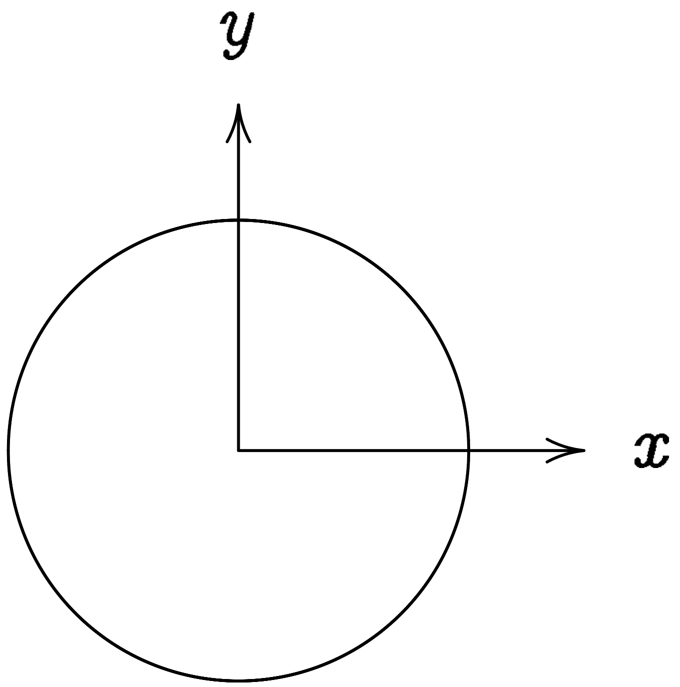
ms

phy":

c

ork."

The clock



This is the curve $x^2 + y^2 = 1$.

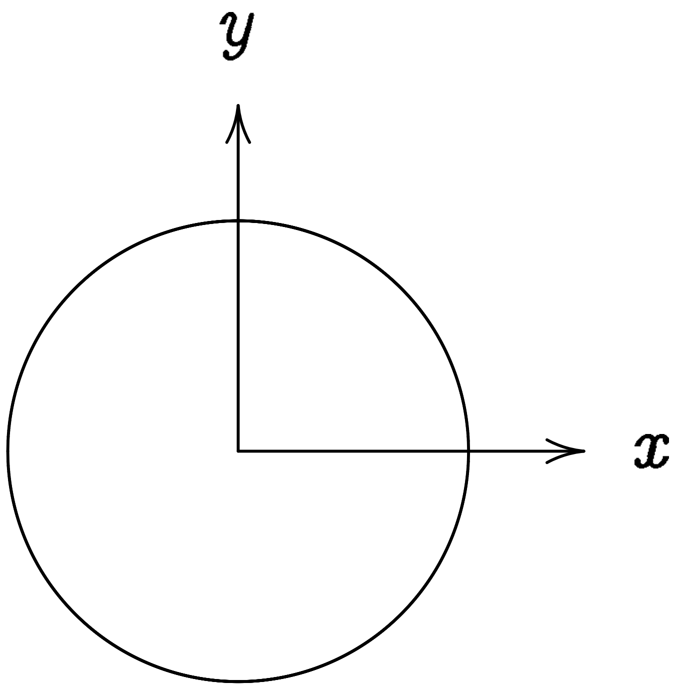
Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

The clock



This is the curve $x^2 + y^2 = 1$.

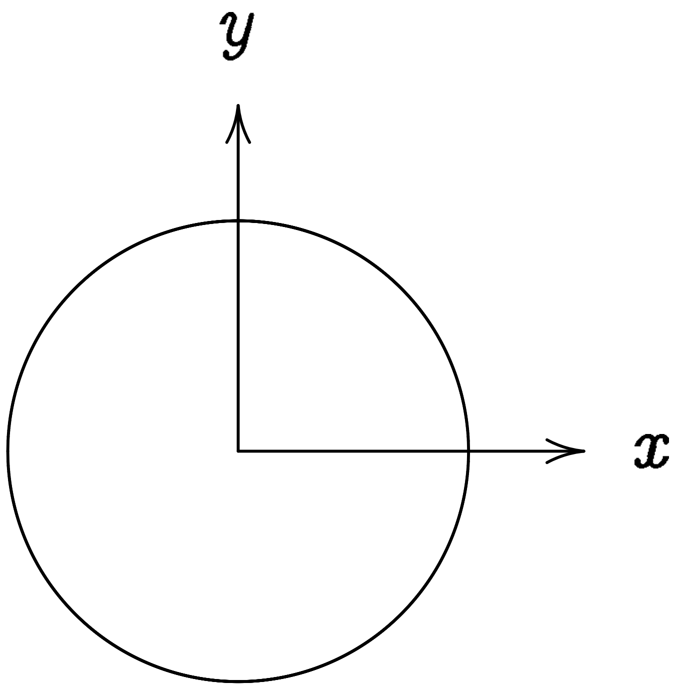
Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

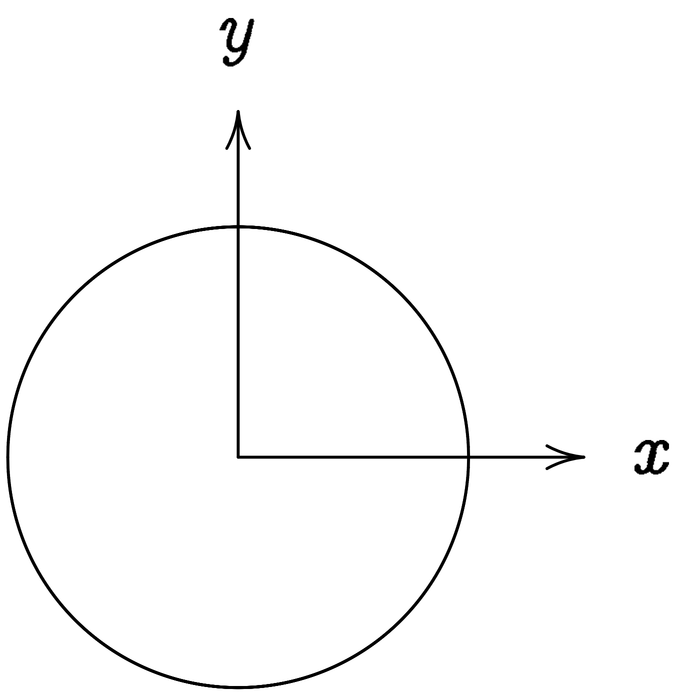
This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

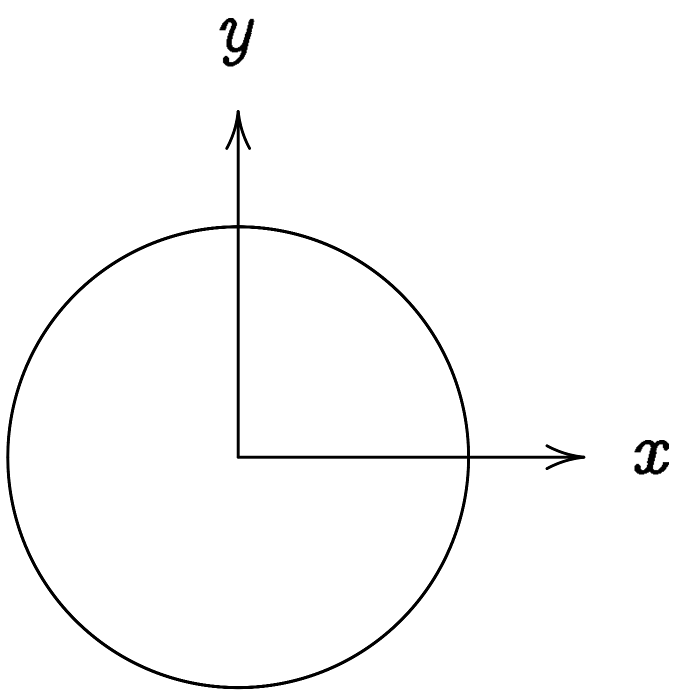
"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

“Elliptic curve” $\neq$ “ellipse.”

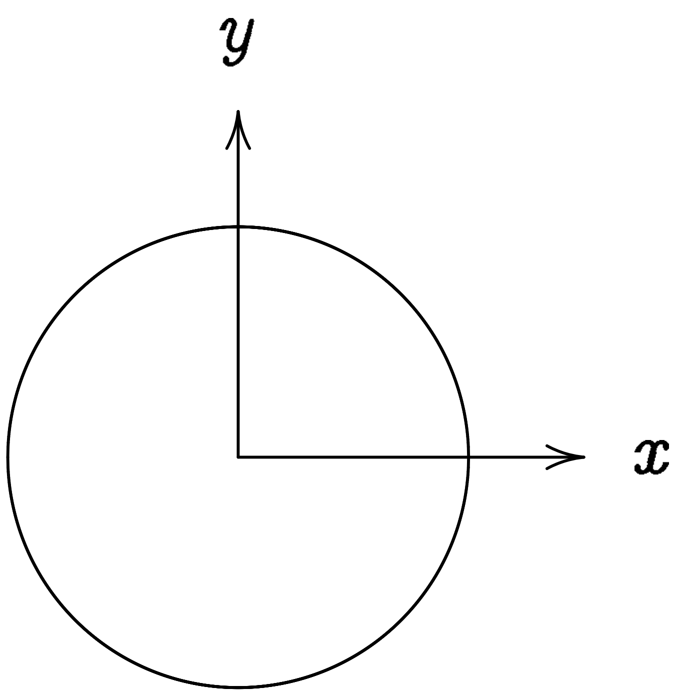
Examples of points on this curve:

$(0, 1) =$ “12:00”.

$(0, -1) =$ “6:00”.

$(1, 0) =$ “3:00”.

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

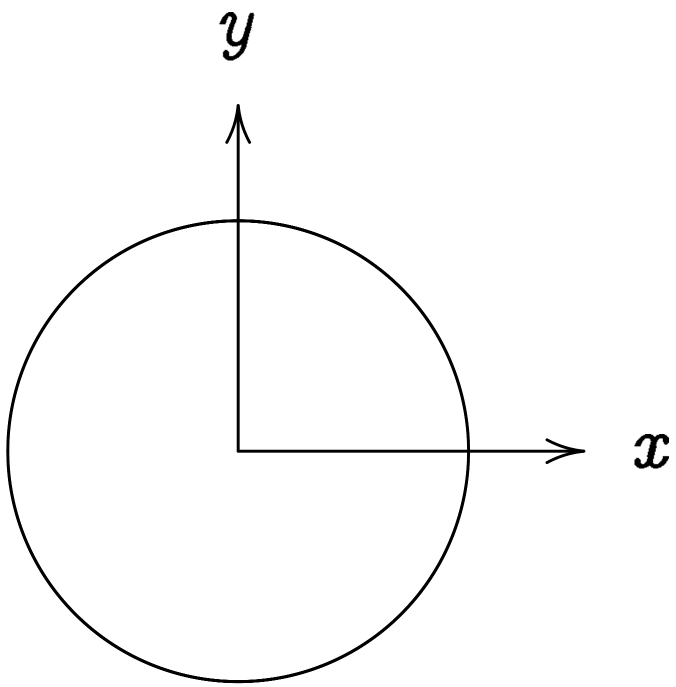
$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

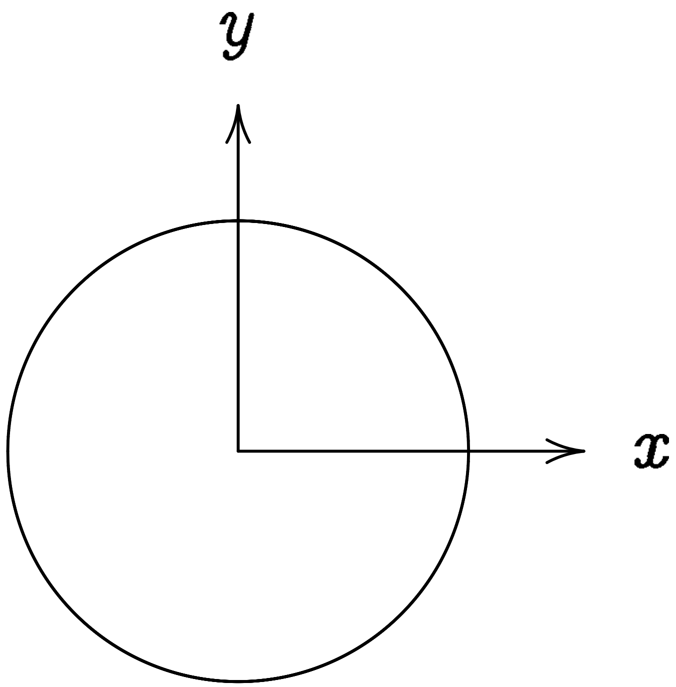
$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

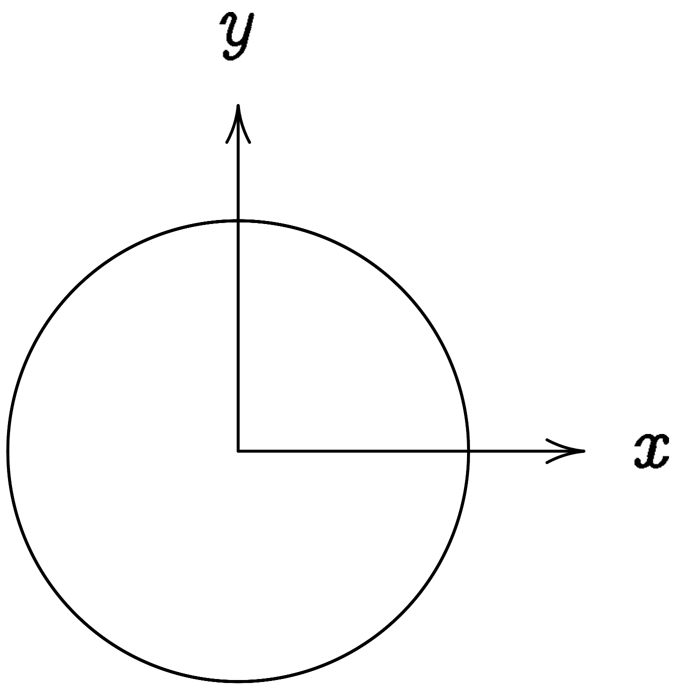
$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

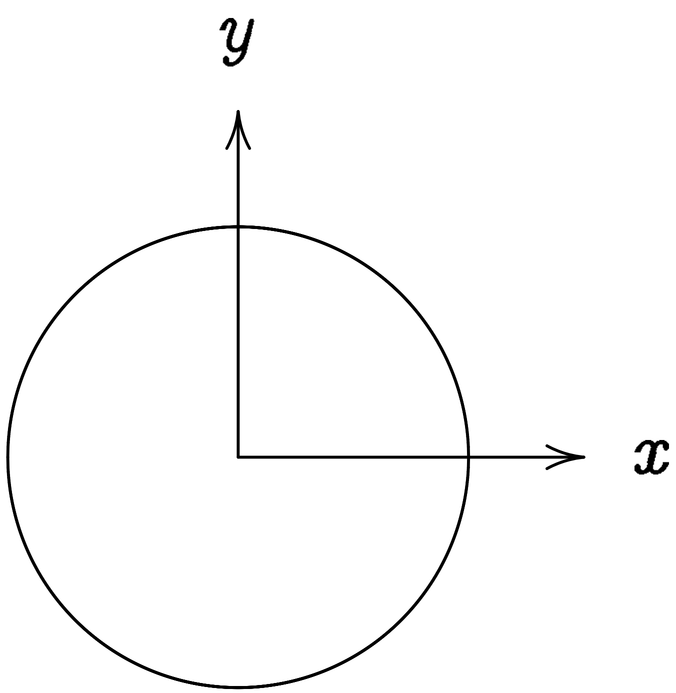
$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

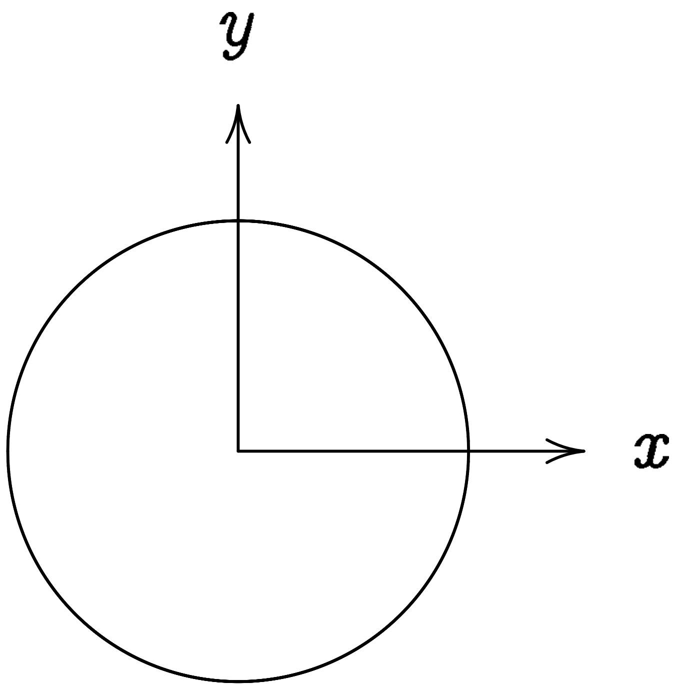
$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

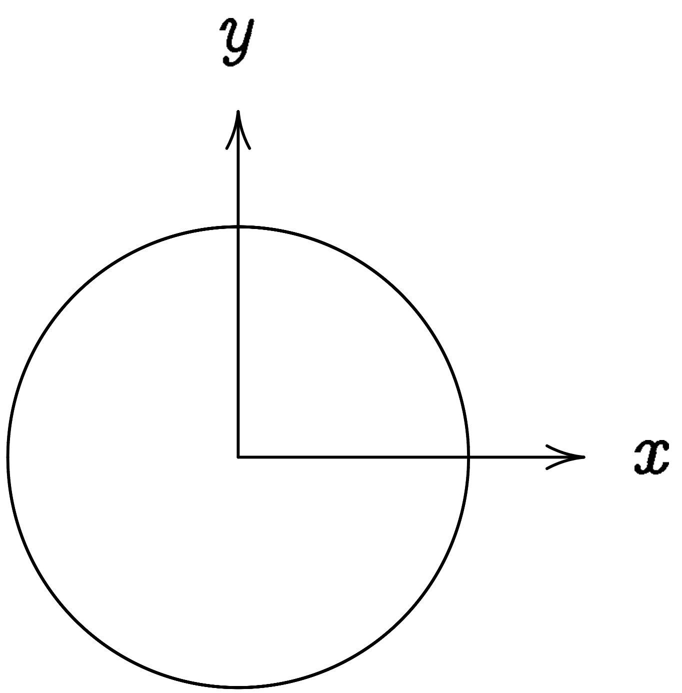
$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

The clock



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

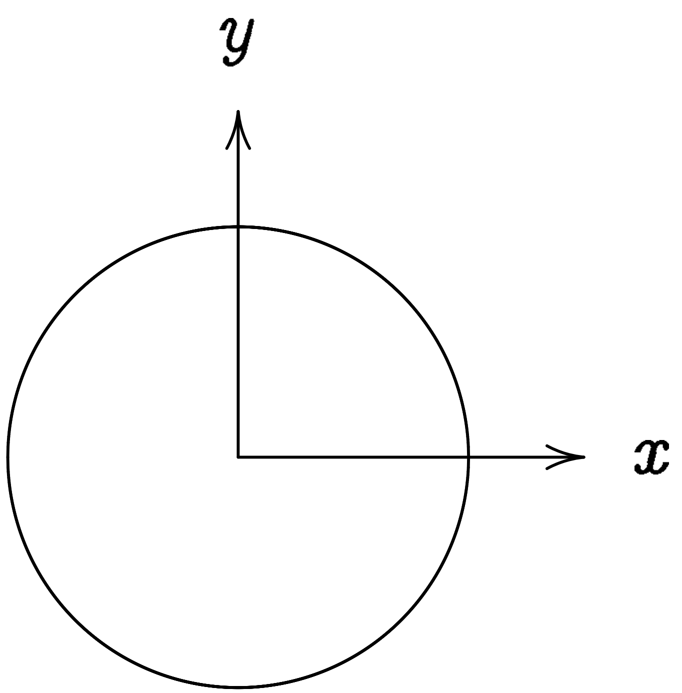
$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

<u>The clock</u>



This is the curve $x^2 + y^2 = 1$.

Warning:

This is *not* an elliptic curve.

"Elliptic curve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

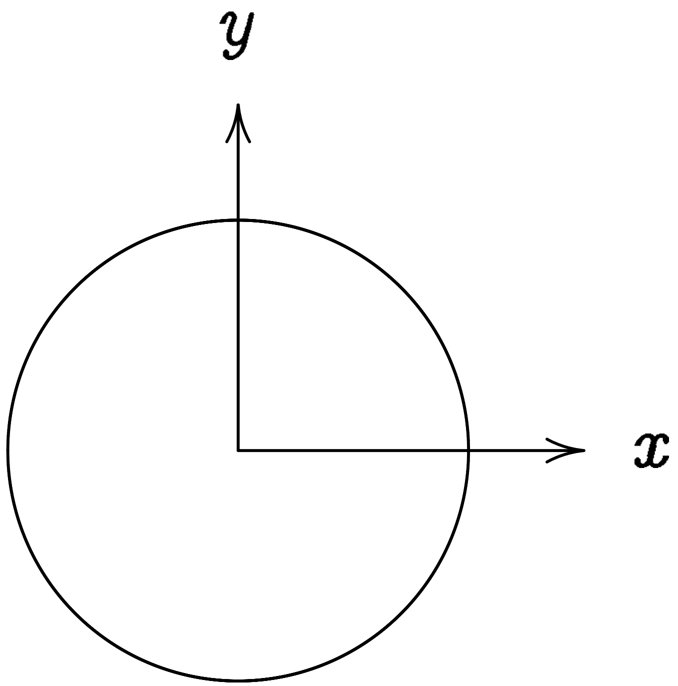
$(3/5, 4/5)$. $(-3/5, 4/5)$.

$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.



curve $x^2 + y^2 = 1$.

an elliptic curve.

urve" $\neq$ "ellipse."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

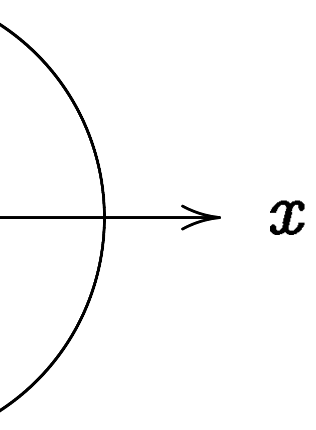
$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition o

$x^2 + y^2 =$

$x = \sin\alpha$,



$y^2 = 1.$

urve.

ose."

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

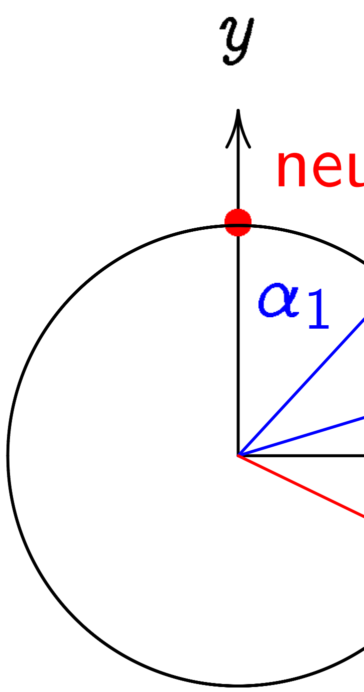
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition on the clock:



$x^2 + y^2 = 1$, parametri

$x = \sin\alpha$, $y = \cos\alpha$.

Examples of points on this curve:

$(0,1) =$ "12:00".

$(0,-1) =$ "6:00".

$(1,0) =$ "3:00".

$(-1,0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

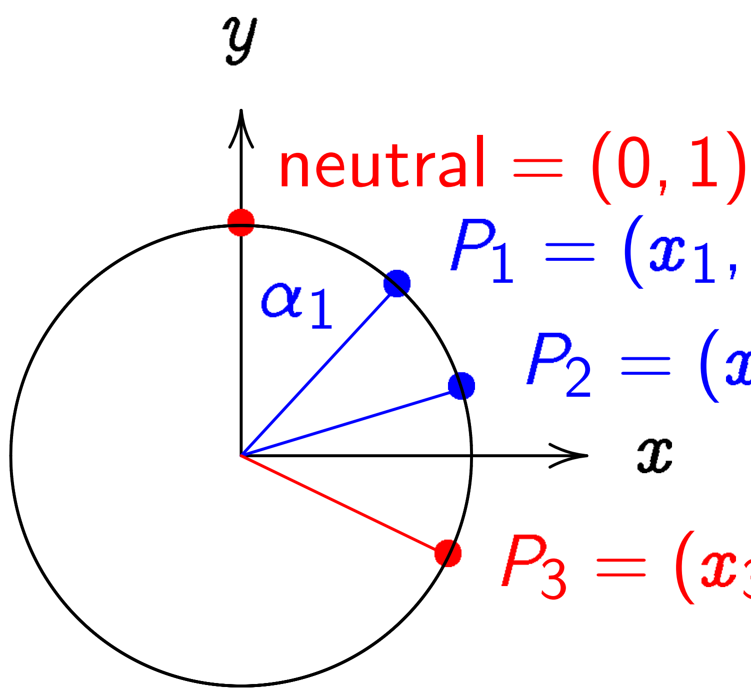
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by

$x = \sin\alpha$, $y = \cos\alpha$.

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

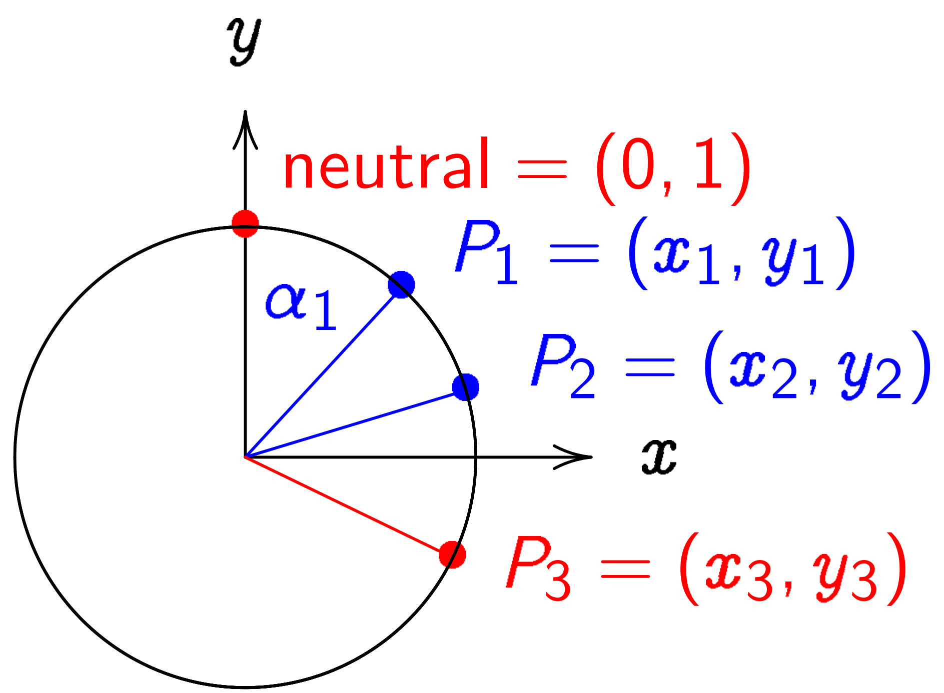
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by

$x = \sin\alpha$, $y = \cos\alpha$.

Examples of points on this curve:

$(0,1) =$ "12:00".

$(0,-1) =$ "6:00".

$(1,0) =$ "3:00".

$(-1,0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

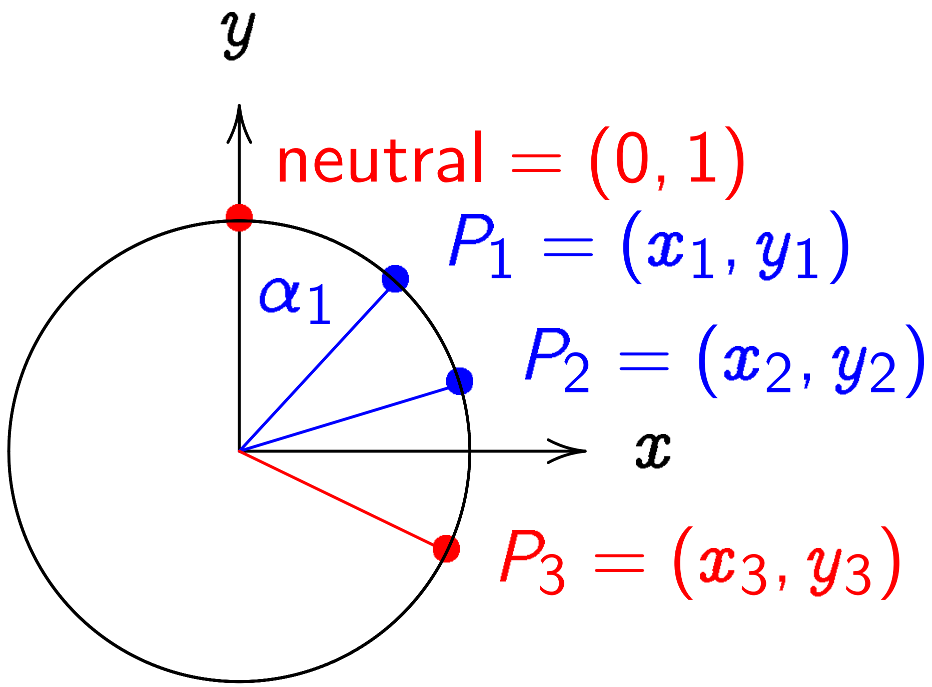
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by

$x = \sin\alpha$, $y = \cos\alpha$. Recall

$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$

Examples of points on this curve:

$(0, 1) =$ "12:00".

$(0, -1) =$ "6:00".

$(1, 0) =$ "3:00".

$(-1, 0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

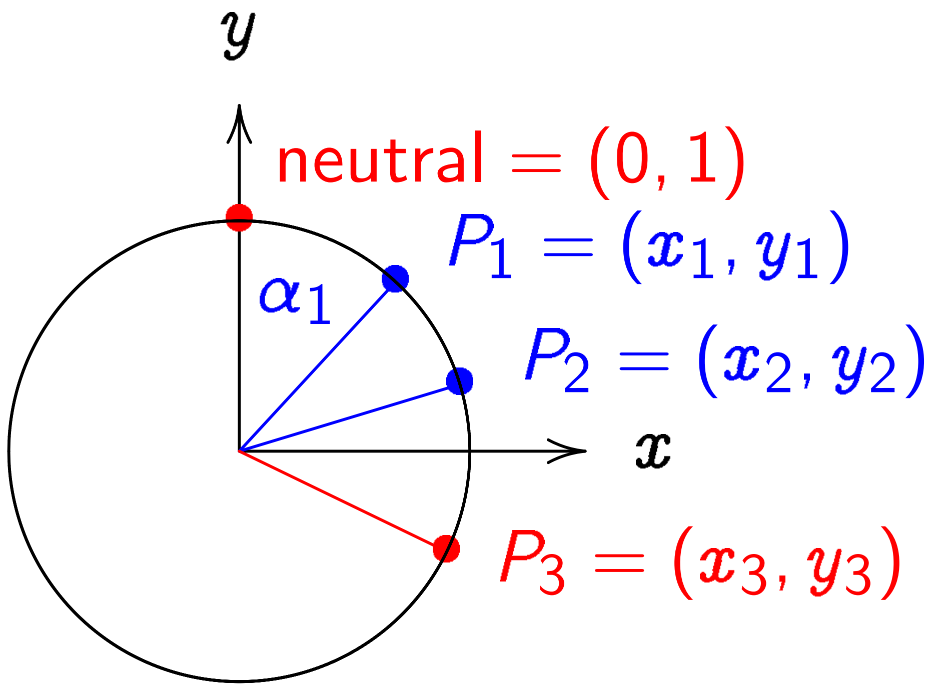
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by

$x = \sin\alpha$, $y = \cos\alpha$. Recall

$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$

$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

Examples of points on this curve:

$(0,1) =$ "12:00".

$(0,-1) =$ "6:00".

$(1,0) =$ "3:00".

$(-1,0) =$ "9:00".

$(\sqrt{3/4}, 1/2) =$ "2:00".

$(1/2, -\sqrt{3/4}) =$ "5:00".

$(-1/2, -\sqrt{3/4}) =$ "7:00".

$(\sqrt{1/2}, \sqrt{1/2}) =$ "1:30".

$(3/5, 4/5)$. $(-3/5, 4/5)$.

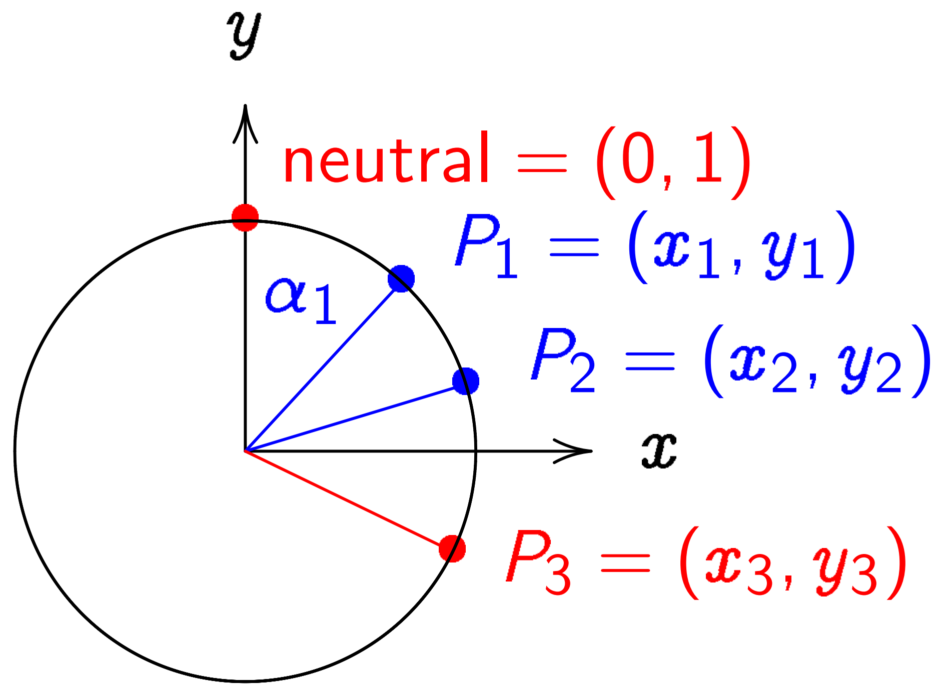
$(3/5, -4/5)$. $(-3/5, -4/5)$.

$(4/5, 3/5)$. $(-4/5, 3/5)$.

$(4/5, -3/5)$. $(-4/5, -3/5)$.

Many more.

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha$, $y = \cos\alpha$. Recall

$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

of points on this curve:

2:00".

"6:00".

:00".

"9:00".

2) = "2:00".

$\overline{3/4}$) = "5:00".

$\sqrt{3/4}$) = "7:00".

$\overline{1/2}$) = "1:30".

$(-3/5, 4/5)$.

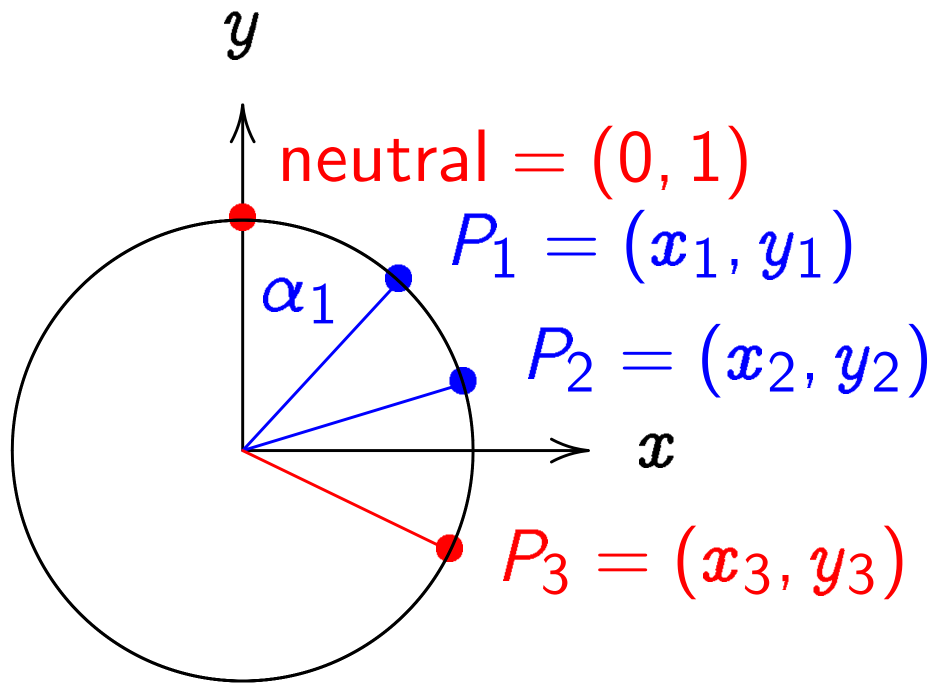
5). $(-3/5, -4/5)$.

$(-4/5, 3/5)$.

5). $(-4/5, -3/5)$.

e.

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by $x = \sin\alpha$, $y = \cos\alpha$. Recall

$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addi

Use Cartes

Addition fo

for the cloc

sum of $(x_1$

$(x_1 y_2 + y_1$

this curve:

”.

00”.

0”.

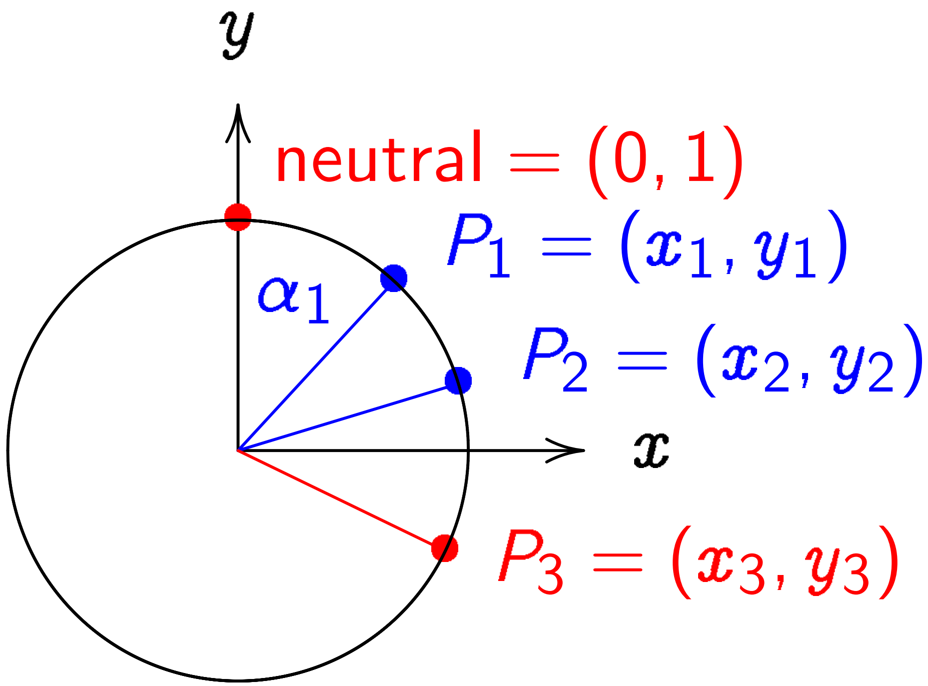
).

4/5).

).

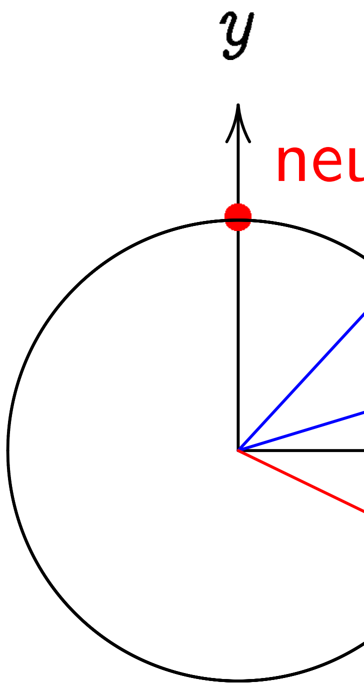
3/5).

Addition on the clock:



$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha$, $y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addition without



Use Cartesian coordinat

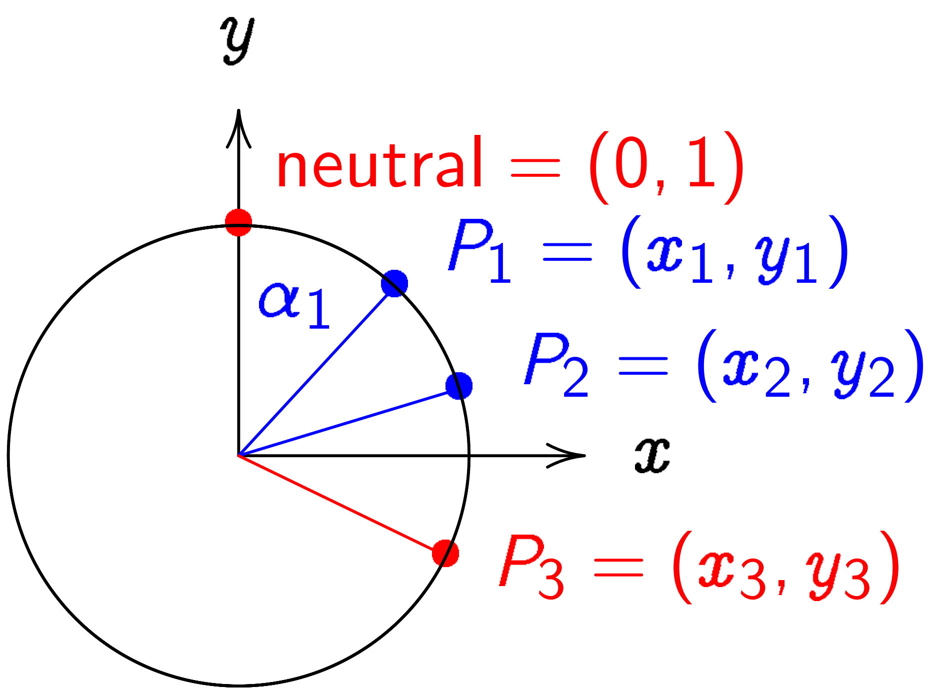
Addition formula

for the clock $x^2 + y^2 =$

sum of (x_1, y_1) and $(x_2$

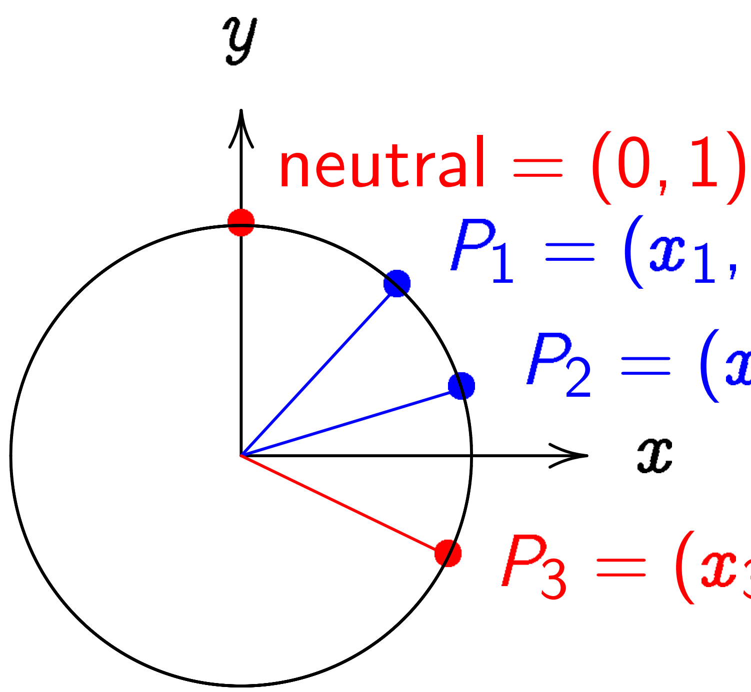
$(x_1 y_2 + y_1 x_2, y_1 y_2 - x$

Addition on the clock:



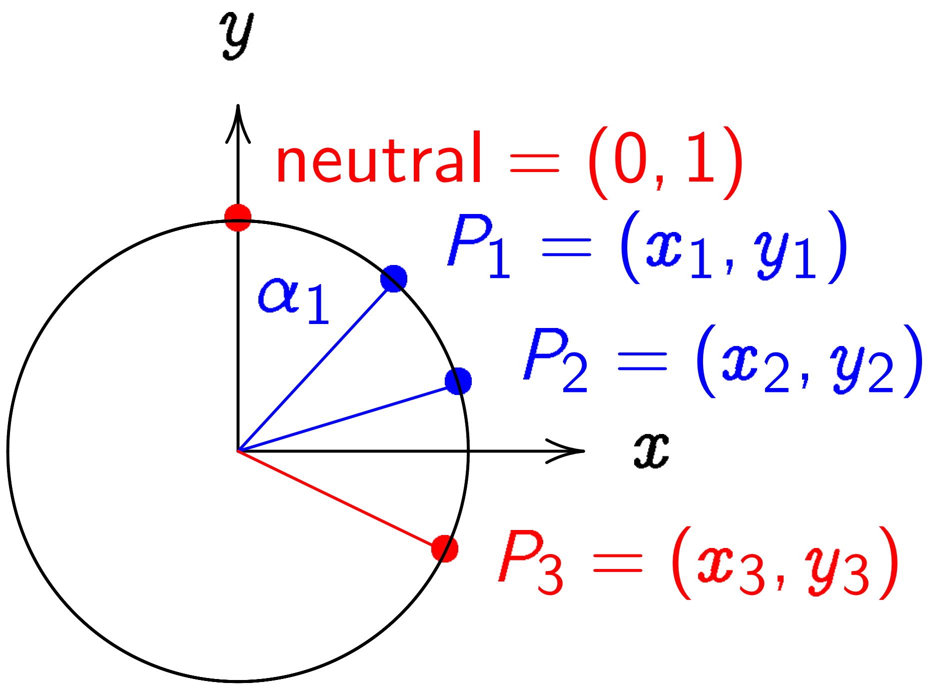
$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha$, $y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addition without sin, cos:



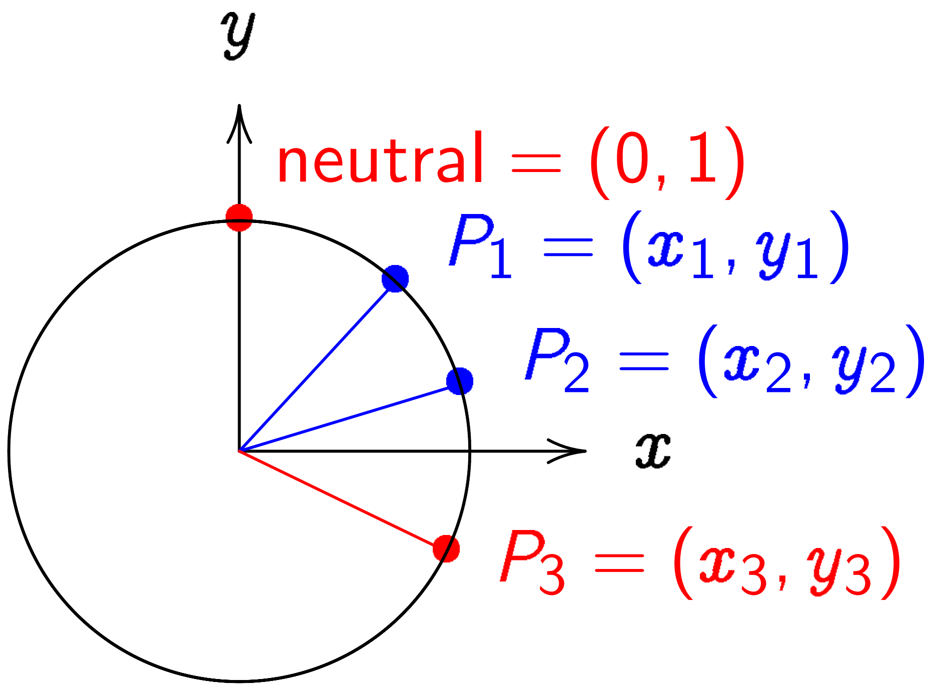
Use Cartesian coordinates for additi
Addition formula
for the clock $x^2 + y^2 = 1$:
sum of (x_1, y_1) and (x_2, y_2) is
$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Addition on the clock:



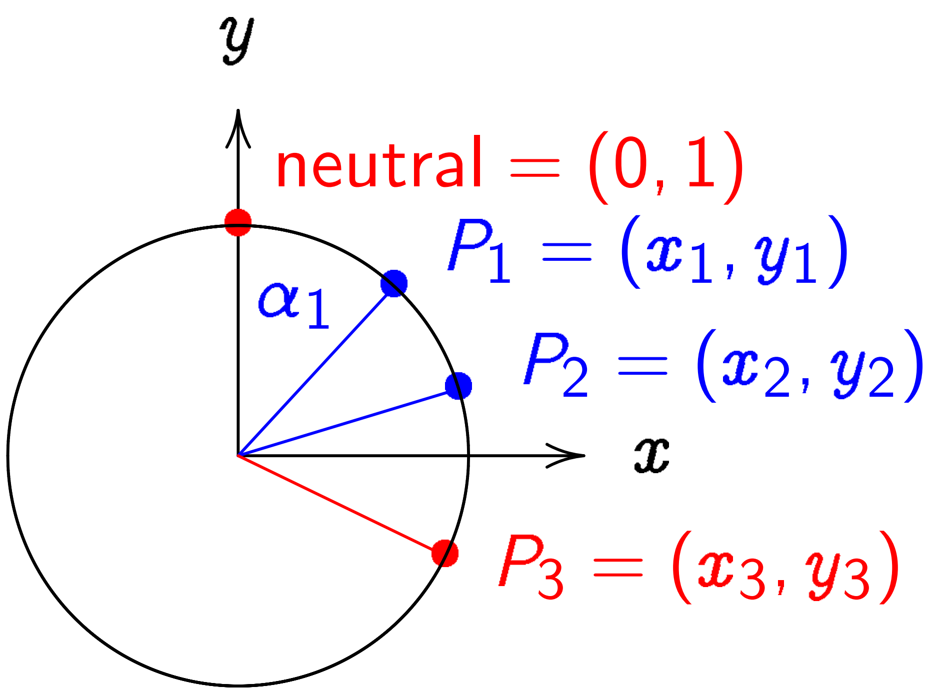
$x^2 + y^2 = 1$, parametrized by
$x = \sin\alpha$, $y = \cos\alpha$. Recall
$(\sin(\alpha_1 + \alpha_2), \cos(\alpha_1 + \alpha_2)) =$
$(\sin\alpha_1 \cos\alpha_2 + \cos\alpha_1 \sin\alpha_2,$
$\cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2)$.

Clock addition without sin, cos:



Use Cartesian coordinates for addition.
Addition formula
for the clock $x^2 + y^2 = 1$:
sum of (x_1, y_1) and (x_2, y_2) is
$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

n the clock:



1, parametrized by

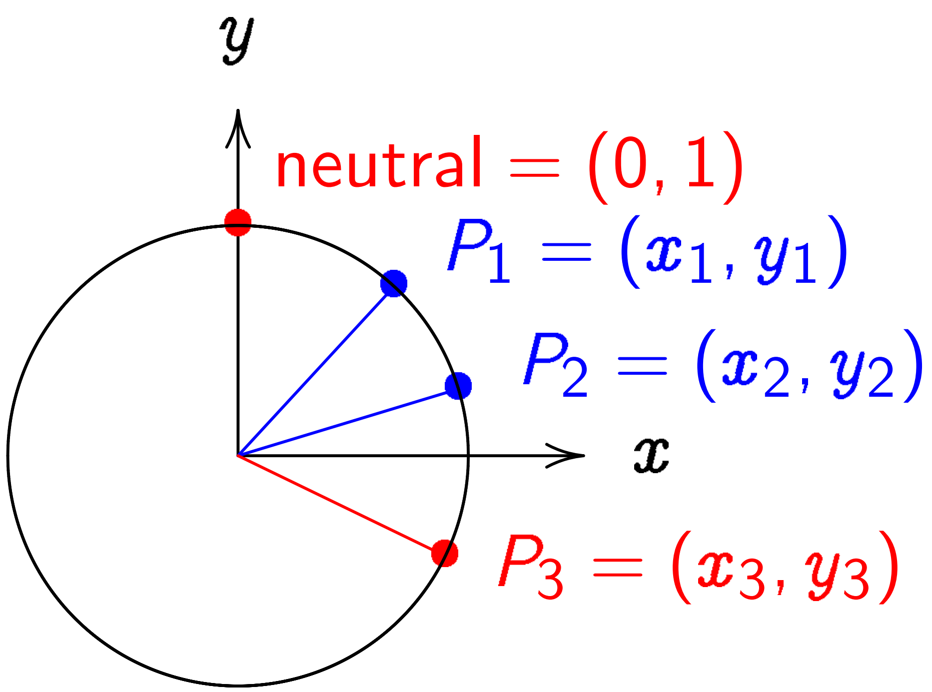
$y = \cos\alpha$. Recall

$\alpha_2), \cos(\alpha_1 + \alpha_2)) =$

$\alpha_2 + \cos\alpha_1 \sin\alpha_2,$

$\alpha_2 - \sin\alpha_1 \sin\alpha_2).$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples c

"2:00" + "

$= (\sqrt{3/4},$

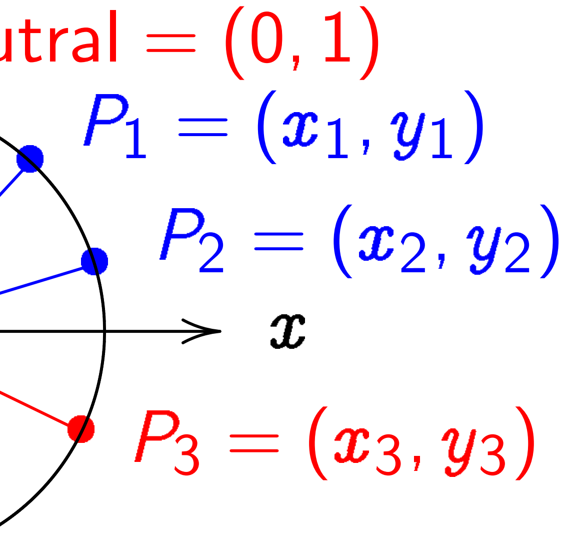
$= (-1/2, -$

"5:00" + "

$= (1/2, -\sqrt{}$

$= (\sqrt{3/4},$

$2\left(\frac{3}{5}, \frac{4}{5}\right) =$



zed by

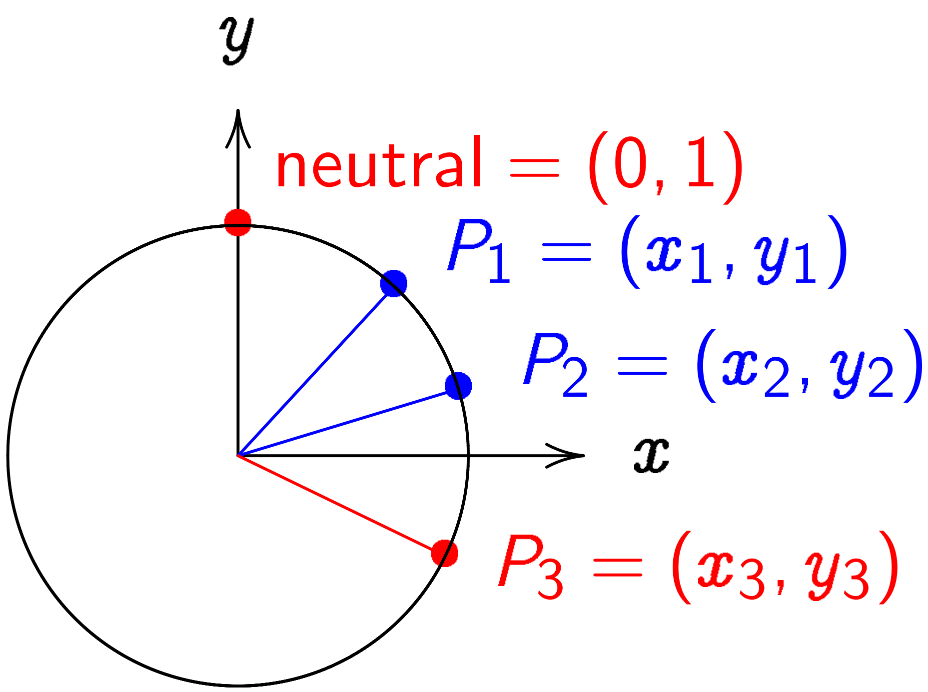
Recall

$-\alpha_2)) =$

$\sin \alpha_2,$

$\sin \alpha_2).$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addi

“2:00” + “5:00”

$= (\sqrt{3/4}, 1/2) + (1/2,$

$= (-1/2, -\sqrt{3/4}) =$ “

“5:00” + “9:00”

$= (1/2, -\sqrt{3/4}) + (-1$

$= (\sqrt{3/4}, 1/2) =$ “2:00

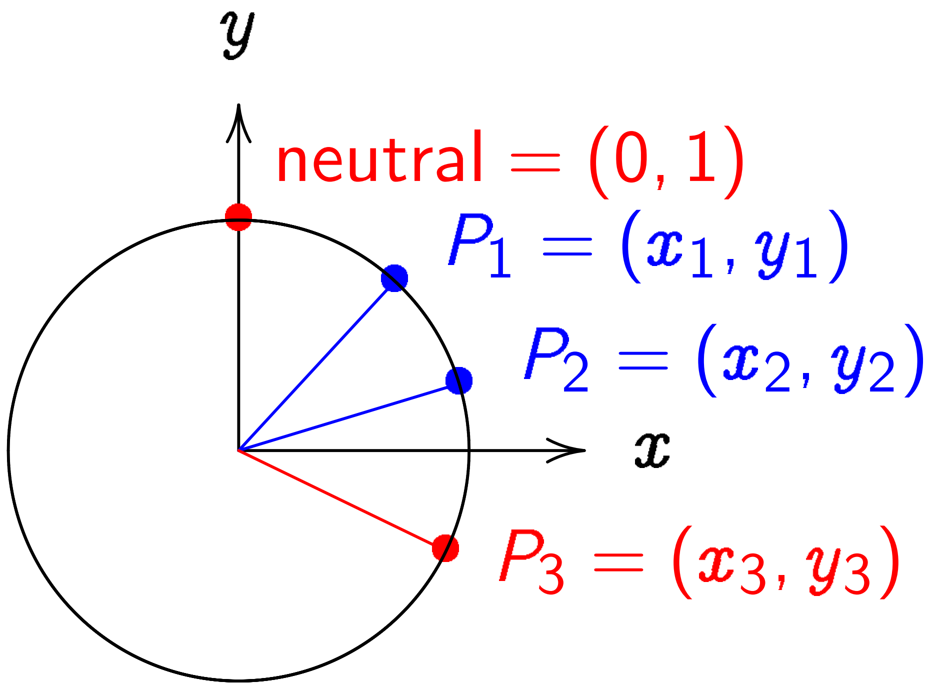
$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right)$

$y_1)$

$x_2, y_2)$

$_3, y_3)$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

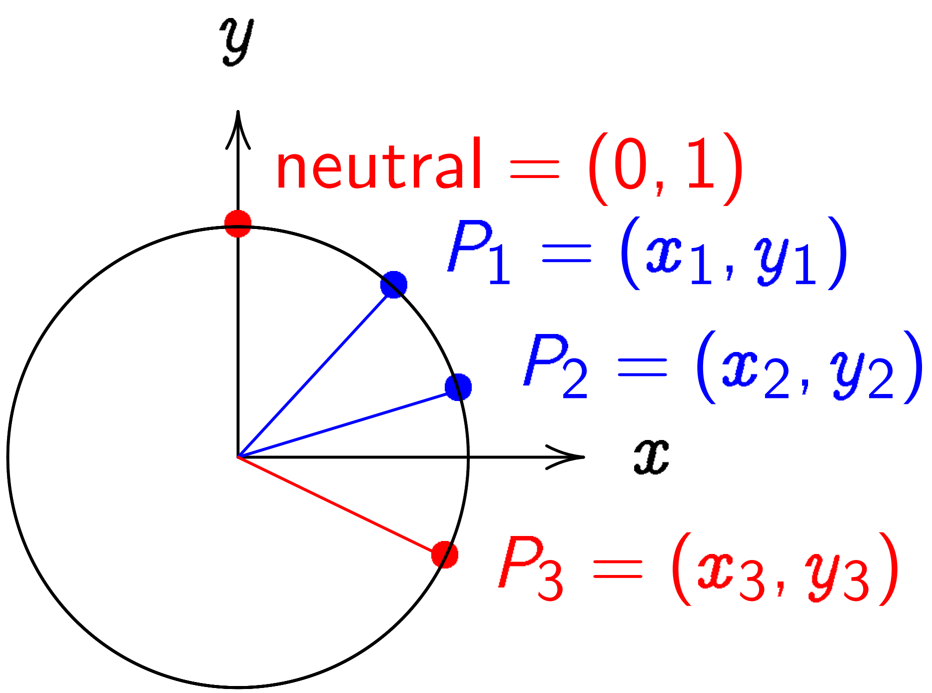
"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

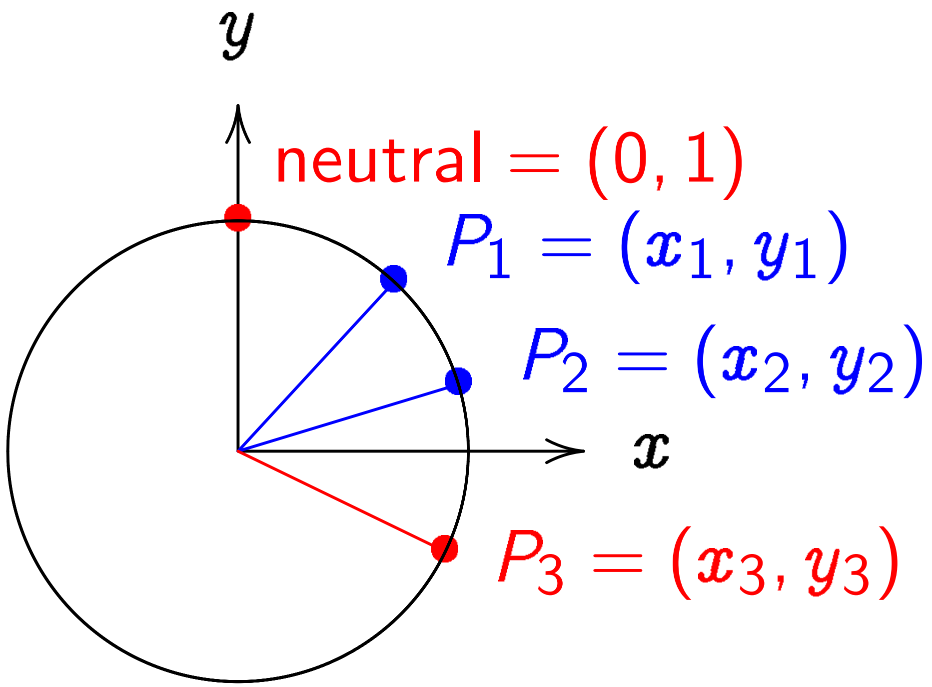
"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

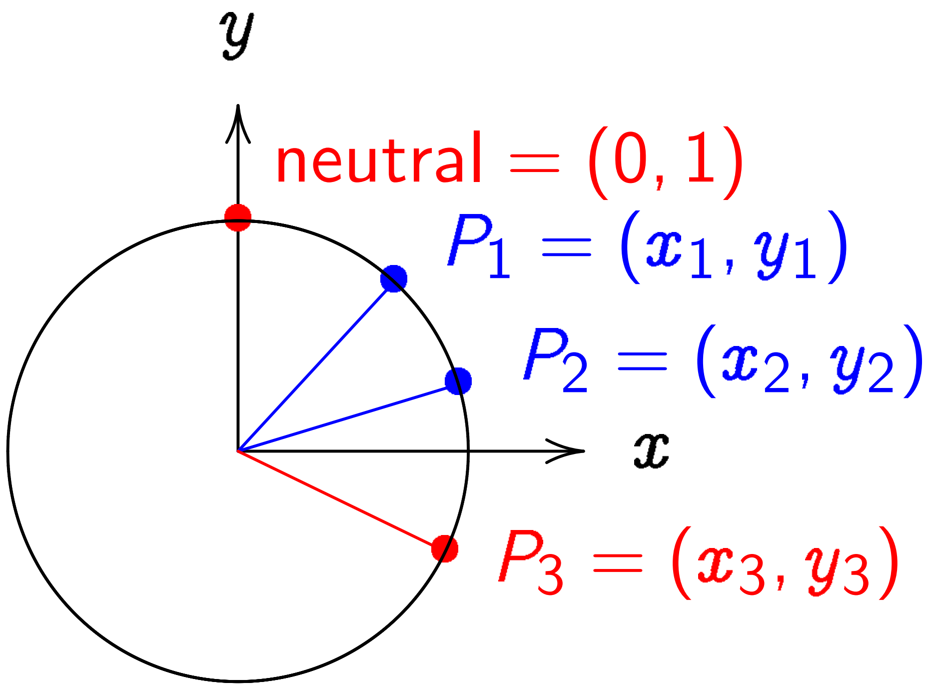
$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

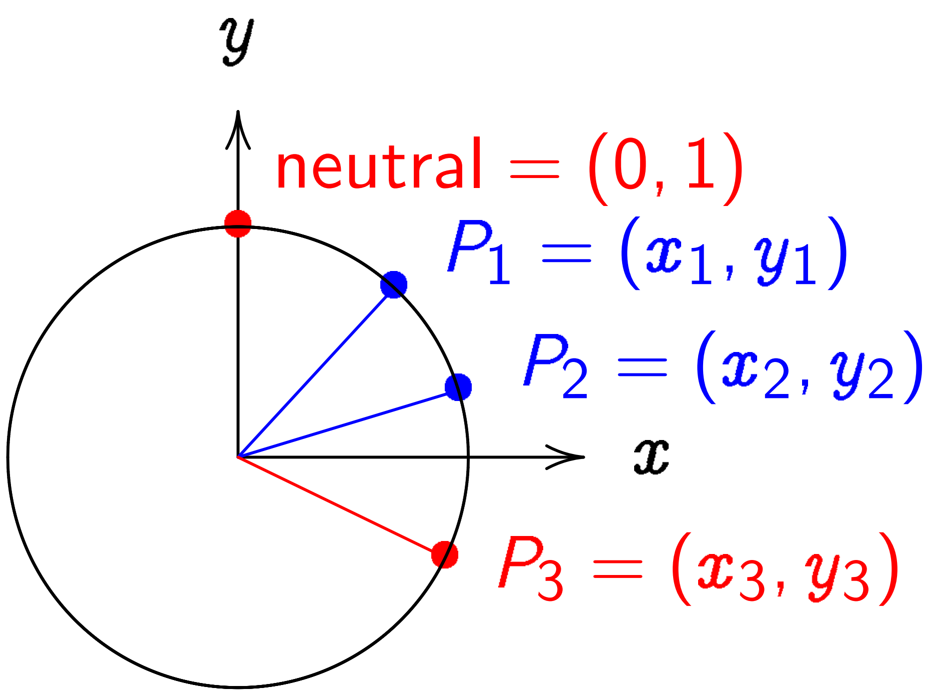
$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

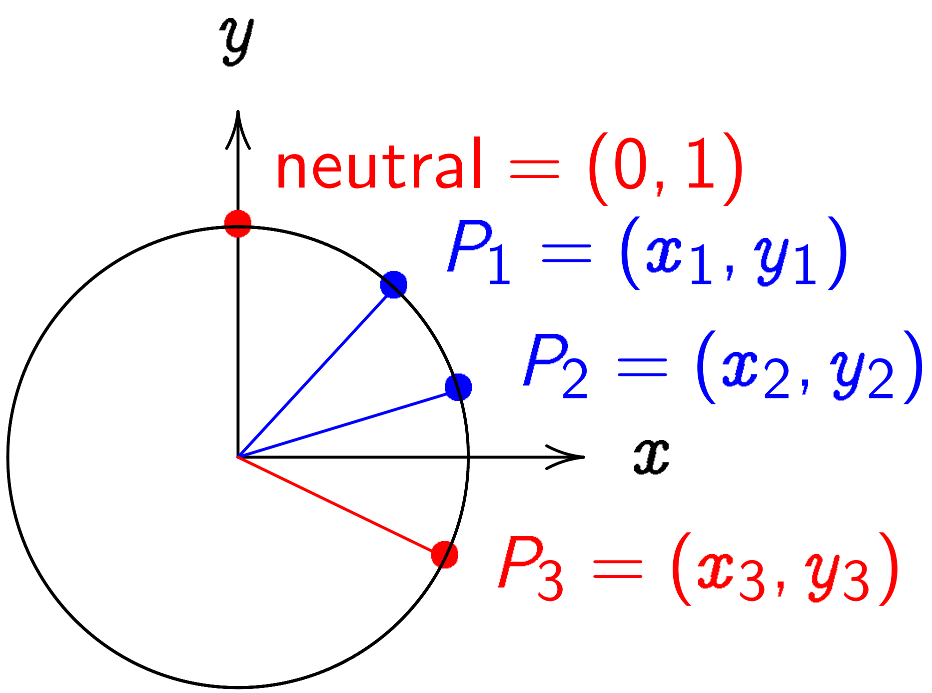
$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) =$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

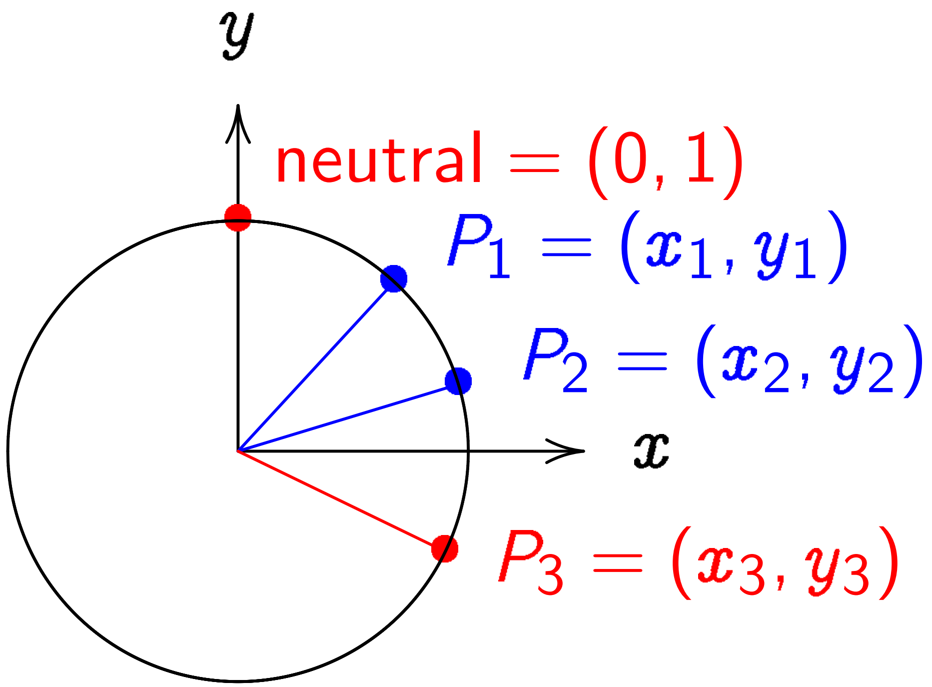
$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

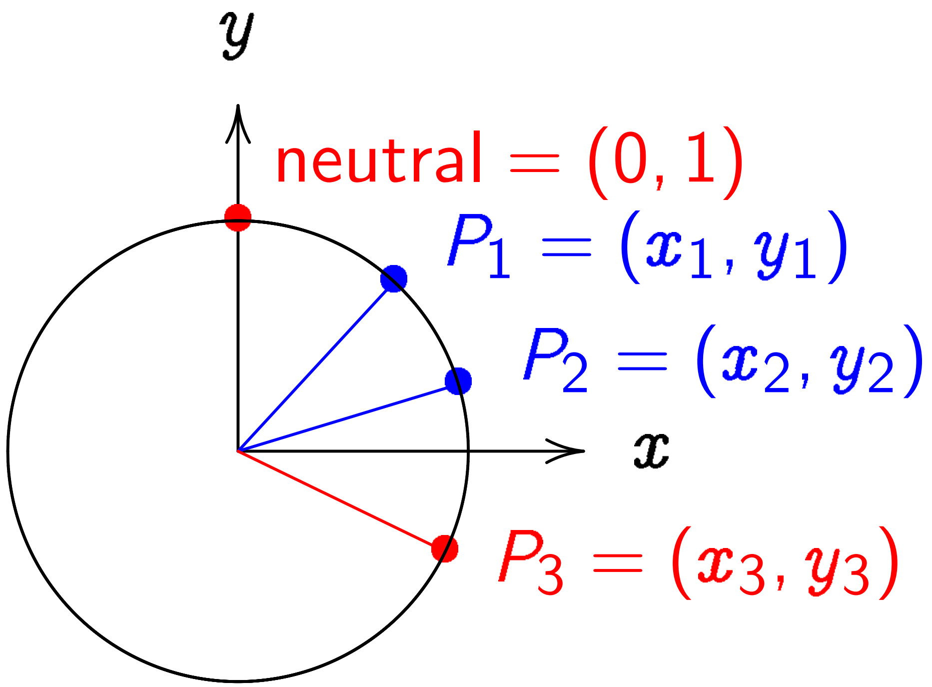
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

$(x_1, y_1) + (-x_1, y_1) =$

Clock addition without sin, cos:



Use Cartesian coordinates for addition.

Addition formula

for the clock $x^2 + y^2 = 1$:

sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2, y_1y_2 - x_1x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

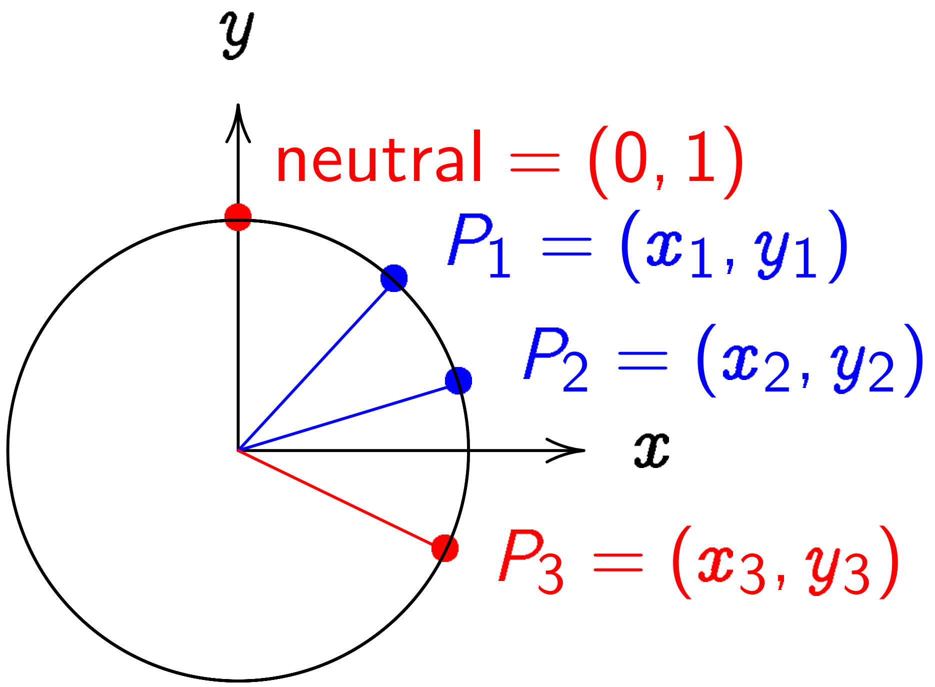
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

tion without sin, cos:



ian coordinates for addition.

ormula

ck $x^2 + y^2 = 1$:

, $y_1)$ and (x_2, y_2) is

$x_2, y_1 y_2 - x_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

Clocks ove

Clock($\mathbf{F}_7$)

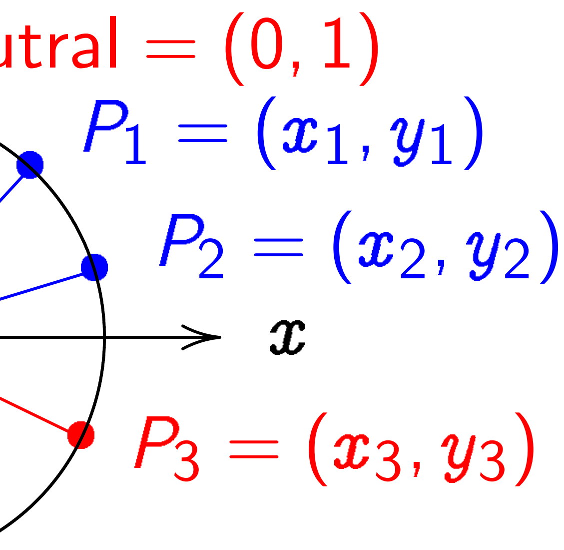
Here $\mathbf{F}_7 =$

$=$

with arithm

e.g. $2 \cdot 5 =$

sin, cos:



tes for addition.

= 1:

$_2, y_2)$ is

$_1 x_2)$.

Examples of clock addition:

"2:00" + "5:00"

$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$

$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"

$= (1/2, -\sqrt{3/4}) + (-1, 0)$

$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

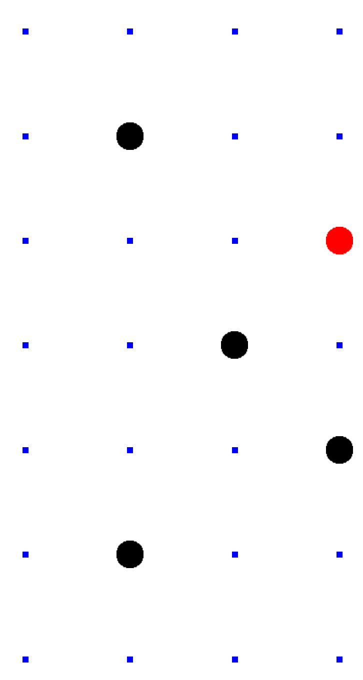
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

Clocks over finite fields



$\mathrm{Clock}(\mathbf{F}_7) = \{(x, y) \in$

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4,$

$= \{0, 1, 2, 3, -$

with arithmetic modulo

e.g. $2 \cdot 5 = 3$ and $3/2 =$

$y_1)$

$x_2, y_2)$

$3, y_3)$

on.

Examples of clock addition:

"2:00" + "5:00"

$$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$$
$$= (-1/2, -\sqrt{3/4}) = \text{“7:00”}.$$

"5:00" + "9:00"

$$= (1/2, -\sqrt{3/4}) + (-1, 0)$$
$$= (\sqrt{3/4}, 1/2) = \text{“2:00”}.$$

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

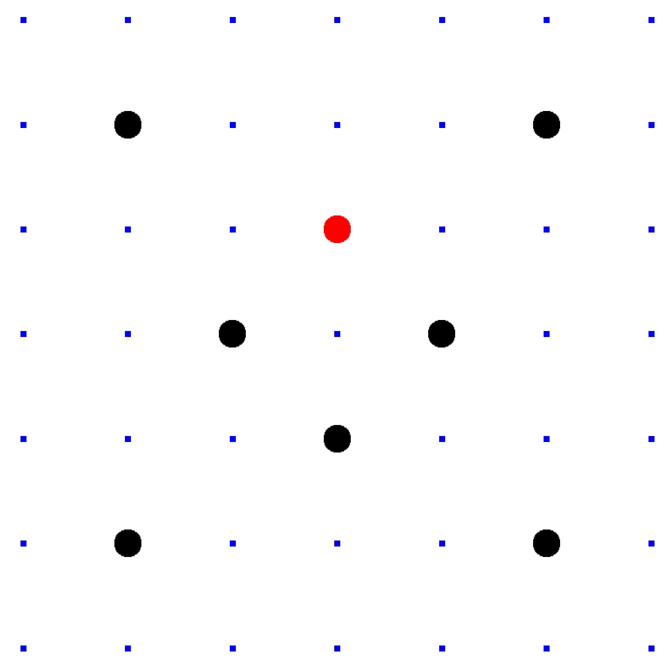
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$$(x_1, y_1) + (0, 1) = (x_1, y_1).$$

$$(x_1, y_1) + (-x_1, y_1) = (0, 1).$$

Clocks over finite fields



$\text{Clock}(\mathbf{F}_7) = \{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2$

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

$= \{0, 1, 2, 3, -3, -2, -1\}$

with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

Examples of clock addition:

"2:00" + "5:00"
$= (\sqrt{3/4}, 1/2) + (1/2, -\sqrt{3/4})$
$= (-1/2, -\sqrt{3/4}) =$ "7:00".

"5:00" + "9:00"
$= (1/2, -\sqrt{3/4}) + (-1, 0)$
$= (\sqrt{3/4}, 1/2) =$ "2:00".

$$2\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{24}{25}, \frac{7}{25}\right).$$

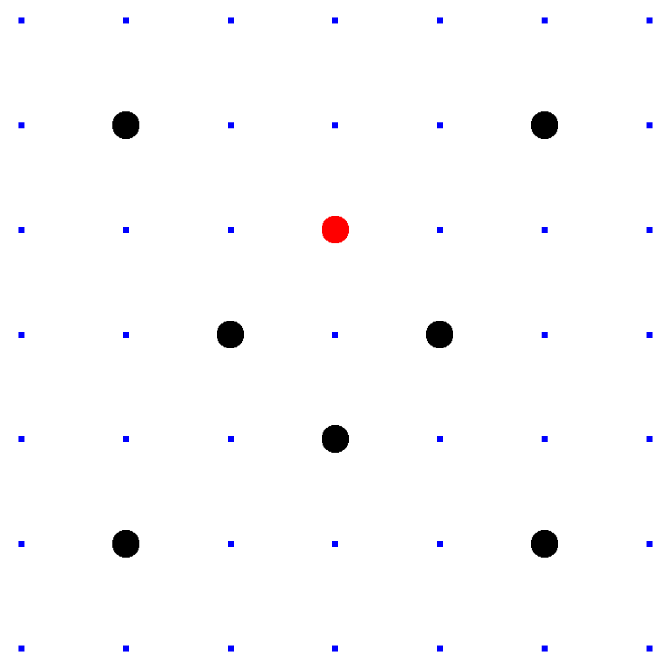
$$3\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{117}{125}, \frac{-44}{125}\right).$$

$$4\left(\frac{3}{5}, \frac{4}{5}\right) = \left(\frac{336}{625}, \frac{-527}{625}\right).$$

$(x_1, y_1) + (0, 1) = (x_1, y_1)$.

$(x_1, y_1) + (-x_1, y_1) = (0, 1)$.

Clocks over finite fields



$\text{Clock}(\mathbf{F}_7) = \{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2+y^2=1\}$.

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$
$= \{0, 1, 2, 3, -3, -2, -1\}$
with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

of clock addition:

5:00”

$1/2) + (1/2, -\sqrt{3/4})$

$-\sqrt{3/4}) =$ “7:00”.

9:00”

$\sqrt{3/4}) + (-1, 0)$

$1/2) =$ “2:00”.

$= \left(\frac{24}{25}, \frac{7}{25}\right).$

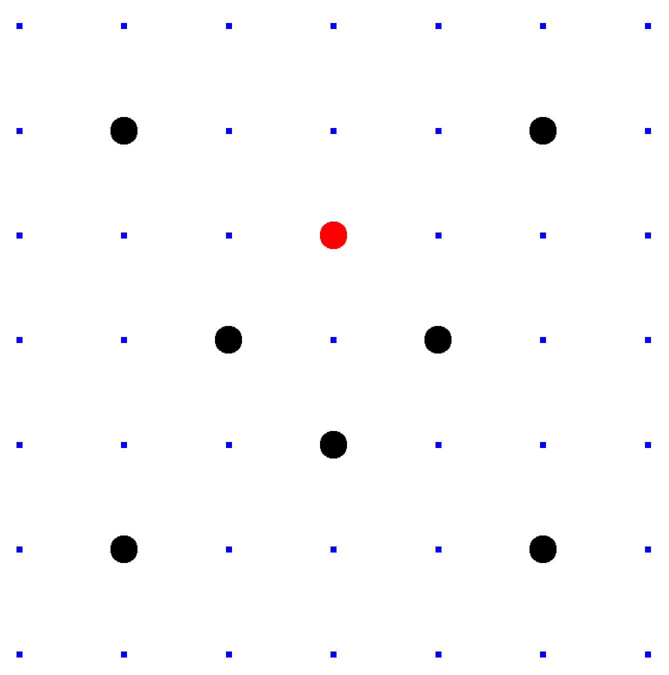
$= \left(\frac{117}{125}, \frac{-44}{125}\right).$

$= \left(\frac{336}{625}, \frac{-527}{625}\right).$

$(0, 1) = (x_1, y_1).$

$(-x_1, y_1) = (0, 1).$

<u>Clocks over finite fields</u>



$\text{Clock}(\mathbf{F}_7) = \{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2+y^2=1\}.$

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

$= \{0, 1, 2, 3, -3, -2, -1\}$

with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for x
...   for
...     if
...
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

tion:

$-\sqrt{3/4})$

7:00".

$1, 0)$

0".

.

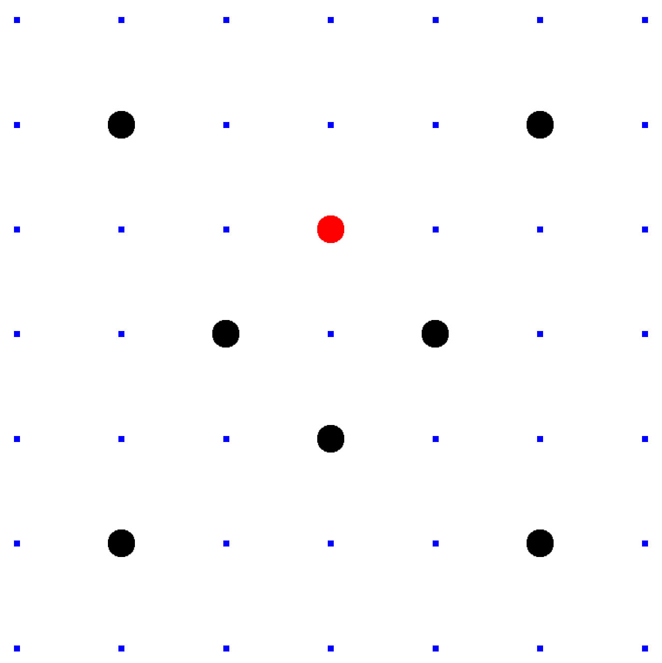
$\frac{4}{5}\Big)$.

$\frac{27}{5}\Big)$.

$y_1)$.

$(0, 1)$.

Clocks over finite fields



$\mathsf{Clock}(\mathbf{F}_7) = \{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2+y^2=1\}$.

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

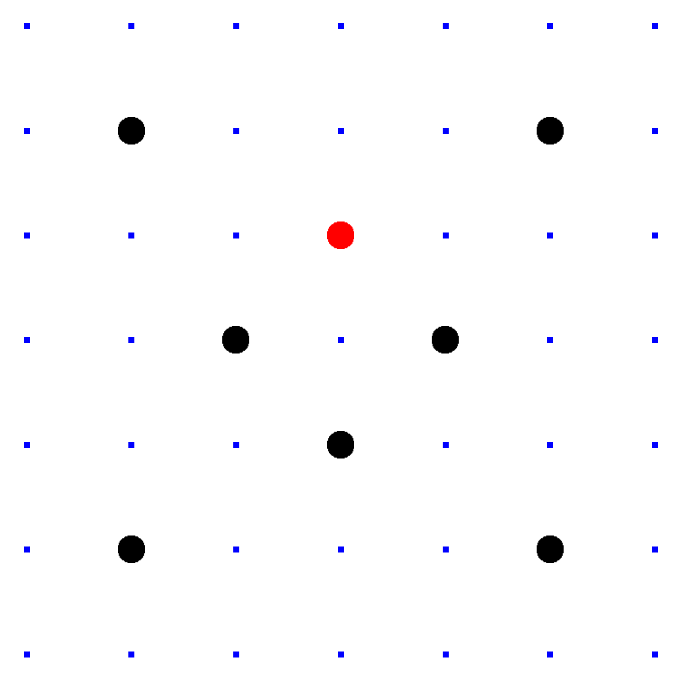
$= \{0, 1, 2, 3, -3, -2, -1\}$

with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for x in range(7)
...   for y in range(
...     if (x*x+y*y)
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

Clocks over finite fields



$\mathrm{Clock}(\mathbf{F}_7) = \left\{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2{+}y^2{=}1\right\}.$

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

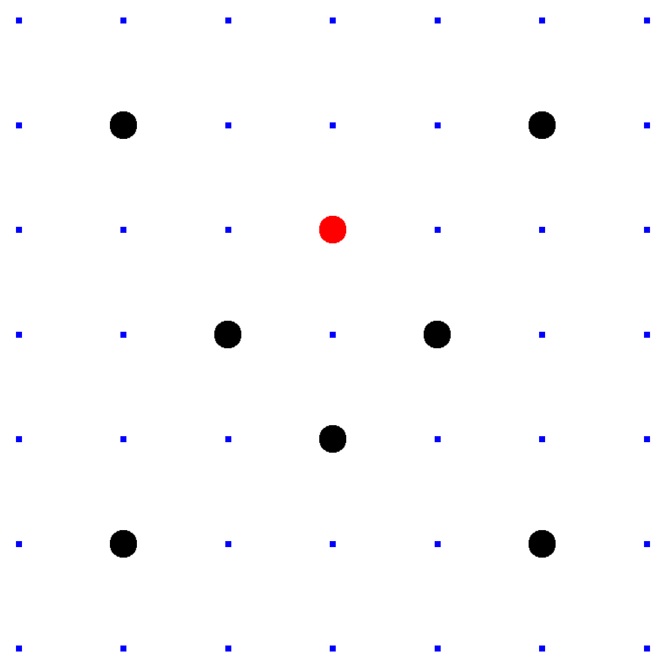
$= \{0, 1, 2, 3, -3, -2, -1\}$

with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

Clocks over finite fields



$\mathrm{Clock}(\mathbf{F}_7) = \{(x, y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2+y^2=1\}$.

Here $\mathbf{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$

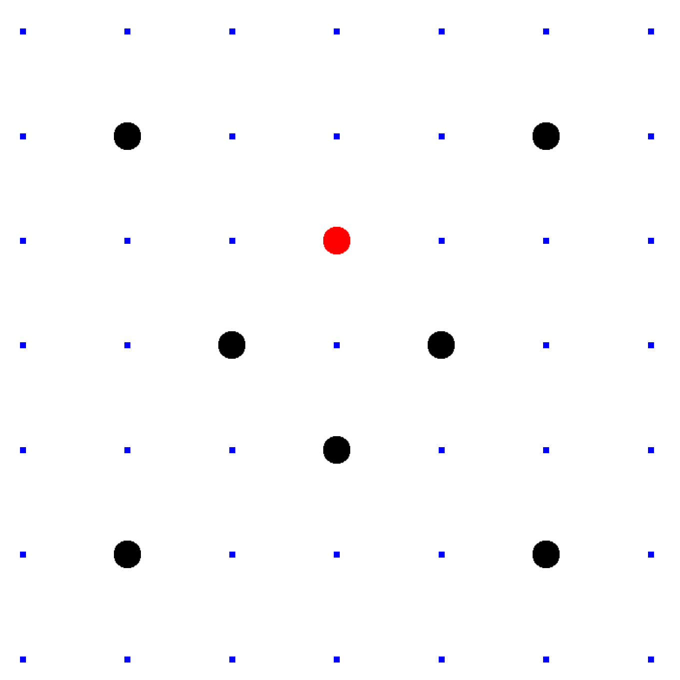
$= \{0, 1, 2, 3, -3, -2, -1\}$

with arithmetic modulo 7.

e.g. $2 \cdot 5 = 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

r finite fields



$= \{(x,y) \in \mathbf{F}_7 \times \mathbf{F}_7 : x^2+y^2=1\}$.

$\{0,1,2,3,4,5,6\}$

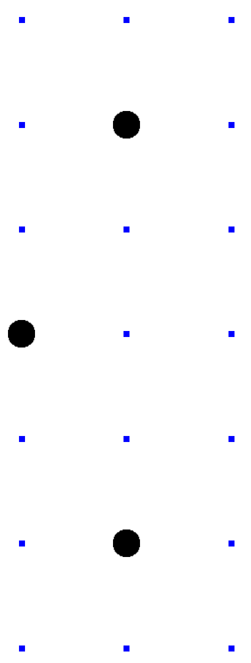
$\{0,1,2,3,-3,-2,-1\}$

netic modulo 7.

$= 3$ and $3/2 = 5$ in $\mathbf{F}_7$.

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

```
>>> class
...   def
...     se
...   def
...     re
...   __re
...
>>> print
2
>>> print
6
>>> print
0
>>> print
3
```



$\mathbf{F}_7 \times \mathbf{F}_7 : x^2{+}y^2{=}1\}$.

$5, 6\}$

$3, -2, -1\}$

$7.$

$= 5$ in $\mathbf{F}_7$.

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

```
>>> class F7:
...   def __init__(se
...     self.int = x
...   def __str__(sel
...     return str(se
...   __repr__ = __st
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

$+y^2=1\}.$

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
>>> for x in range(7):
...   for y in range(7):
...     if (x*x+y*y) % 7 == 1:
...       print (x,y)
...
(0, 1)
(0, 6)
(1, 0)
(2, 2)
(2, 5)
(5, 2)
(5, 5)
(6, 0)
>>>
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
 in range(7):
 y in range(7):
f (x*x+y*y) % 7 == 1:
 print (x,y)
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
>>> F7.__e
...   lamb
>>>
>>> print
True
>>> print
True
>>> print
True
>>> print
False
>>> print
False
>>> print
False
```

```
):
(7):
 % 7 == 1:
)
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.i
>>>
>>> print F7(7) == F7
True
>>> print F7(10) == F
True
>>> print F7(-3) == F
True
>>> print F7(0) == F7
False
>>> print F7(0) == F7
False
>>> print F7(0) == F7
False
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
>>> class F7:
...   def __init__(self,x):
...     self.int = x % 7
...   def __str__(self):
...     return str(self.int)
...   __repr__ = __str__
...
>>> print F7(2)
2
>>> print F7(6)
6
>>> print F7(7)
0
>>> print F7(10)
3
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
F7:
__init__(self,x):
elf.int = x % 7
__str__(self):
eturn str(self.int)
epr__ = __str__

F7(2)

F7(6)

F7(7)

F7(10)
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
>>> F7.__a
...   lamb
>>> F7.__s
...   lamb
>>> F7.__m
...   lamb
>>>
>>> print
0
>>> print
4
>>> print
3
>>>
```

```
elf,x):
 % 7
lf):
elf.int)
tr__
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(
>>> F7.__sub__ = \
...   lambda a,b: F7(
>>> F7.__mul__ = \
...   lambda a,b: F7(
>>>
>>> print F7(2) + F7(
0
>>> print F7(2) - F7(
4
>>> print F7(2) * F7(
3
>>>
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(a.int + b.i
>>> F7.__sub__ = \
...   lambda a,b: F7(a.int - b.i
>>> F7.__mul__ = \
...   lambda a,b: F7(a.int * b.i
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

```
>>> F7.__eq__ = \
...   lambda a,b: a.int == b.int
>>>
>>> print F7(7) == F7(0)
True
>>> print F7(10) == F7(3)
True
>>> print F7(-3) == F7(4)
True
>>> print F7(0) == F7(1)
False
>>> print F7(0) == F7(2)
False
>>> print F7(0) == F7(3)
False
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(a.int + b.int)
>>> F7.__sub__ = \
...   lambda a,b: F7(a.int - b.int)
>>> F7.__mul__ = \
...   lambda a,b: F7(a.int * b.int)
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

```
eq__ = \
bda a,b: a.int == b.int

 F7(7) == F7(0)

 F7(10) == F7(3)

 F7(-3) == F7(4)

 F7(0) == F7(1)

 F7(0) == F7(2)

 F7(0) == F7(3)
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(a.int + b.int)
>>> F7.__sub__ = \
...   lambda a,b: F7(a.int - b.int)
>>> F7.__mul__ = \
...   lambda a,b: F7(a.int * b.int)
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

Larger exa

```
p = 10000
class Fp:
  ...

def clocka
  x1,y1 =
  x2,y2 =
  x3 = x1*
  y3 = y1*
  return x
```

```
int == b.int

7(0)

F7(3)

F7(4)

7(1)

7(2)

7(3)
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(a.int + b.int)
>>> F7.__sub__ = \
...   lambda a,b: F7(a.int - b.int)
>>> F7.__mul__ = \
...   lambda a,b: F7(a.int * b.int)
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

Larger example: Clock(

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(a.int + b.int)
>>> F7.__sub__ = \
...   lambda a,b: F7(a.int - b.int)
>>> F7.__mul__ = \
...   lambda a,b: F7(a.int * b.int)
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

Larger example: Clock($\mathbf{F}_{1000003}$).

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> F7.__add__ = \
...   lambda a,b: F7(a.int + b.int)
>>> F7.__sub__ = \
...   lambda a,b: F7(a.int - b.int)
>>> F7.__mul__ = \
...   lambda a,b: F7(a.int * b.int)
>>>
>>> print F7(2) + F7(5)
0
>>> print F7(2) - F7(5)
4
>>> print F7(2) * F7(5)
3
>>>
```

Larger example: Clock($\mathbf{F}_{1000003}$).

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
add__ = \
bda a,b: F7(a.int + b.int)
sub__ = \
bda a,b: F7(a.int - b.int)
mul__ = \
bda a,b: F7(a.int * b.int)

 F7(2) + F7(5)

 F7(2) - F7(5)

 F7(2) * F7(5)
```

Larger example: Clock($\mathbf{F}_{1000003}$).

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> P = (F
>>> P2 = c
>>> print
(4000, 7)
>>> P3 = c
>>> print
(15000, 26
>>> P4 = c
>>> P5 = c
>>> P6 = c
>>> print
(780000, 1
>>> print
(780000, 1
>>>
```

```
(a.int + b.int)

(a.int - b.int)

(a.int * b.int)

(5)

(5)

(5)
```

Larger example: Clock($\mathbf{F}_{1000003}$).

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> P = (Fp(1000),Fp(
>>> P2 = clockadd(P,P
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,
>>> P5 = clockadd(P4,
>>> P6 = clockadd(P5,
>>> print P6
(780000, 1351)
>>> print clockadd(P3
(780000, 1351)
>>>
```

nt)

nt)

nt)

Larger example: Clock($\mathbf{F}_{1000003}$).

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

Larger example: Clock($\mathbf{F}_{1000003}$).

```
p = 1000003
class Fp:
  ...

def clockadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = x1*y2+y1*x2
  y3 = y1*y2-x1*x2
  return x3,y3
```

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

mple: Clock($\mathbf{F}_{1000003}$).

```
03
```

```
add(P1,P2):
 P1
 P2
*y2+y1*x2
*y2-x1*x2
x3,y3
```

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

```
>>> def sc
...   if n
...   if n
...   Q =
...   Q =
...   if n
...   retu
...
>>> n = ou
>>> scalar
(947472, 7
>>>
```

Can you fig

$(\mathbf{F}_{1000003})$.

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

```
>>> def scalarmult(n,
...   if n == 0: retu
...   if n == 1: retu
...   Q = scalarmult(
...   Q = clockadd(Q,
...   if n % 2: Q = c
...   return Q
...
>>> n = oursixdigitse
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),F
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

```
>>> P = (Fp(1000),Fp(2))
>>> P2 = clockadd(P,P)
>>> print P2
(4000, 7)
>>> P3 = clockadd(P2,P)
>>> print P3
(15000, 26)
>>> P4 = clockadd(P3,P)
>>> P5 = clockadd(P4,P)
>>> P6 = clockadd(P5,P)
>>> print P6
(780000, 1351)
>>> print clockadd(P3,P3)
(780000, 1351)
>>>
```

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),Fp(1))
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q)
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

```
Fp(1000),Fp(2))
clockadd(P,P)
 P2

clockadd(P2,P)
 P3
6)
clockadd(P3,P)
clockadd(P4,P)
clockadd(P5,P)
 P6
1351)
 clockadd(P3,P3)
1351)
```

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),Fp(1))
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q)
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

Clock cryp

The "Clock

Standardiz
and **base p**

Alice choos
Alice comp

Bob choos
Bob compu

Alice comp
Bob compu
They use t
to encrypt

```
(2))
P)
,P)
,P)
,P)
,P)
3,P3)
```

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),Fp(1))
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q)
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

Clock cryptography

The “Clock Diffie–Hell

Standardize a large prin
and **base point** (x, y)

Alice chooses big secret
Alice computes her pub

Bob chooses big secret
Bob computes his publi

Alice computes $a(b(x, y$
Bob computes $b(a(x, y$
They use this shared se
to encrypt with AES-G

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),Fp(1))
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q)
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

Clock cryptography

The "Clock Diffie–Hellman protocol

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x,$

Bob chooses big secret b.
Bob computes his public key $b(x, y)$

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.

```
>>> def scalarmult(n,P):
...   if n == 0: return (Fp(0),Fp(1))
...   if n == 1: return P
...   Q = scalarmult(n//2,P)
...   Q = clockadd(Q,Q)
...   if n % 2: Q = clockadd(P,Q)
...   return Q
...
>>> n = oursixdigitsecret
>>> scalarmult(n,P)
(947472, 736284)
>>>
```

Can you figure out our secret n?

Clock cryptography

The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.

```
calarmult(n,P):
n == 0: return (Fp(0),Fp(1))
n == 1: return P
 scalarmult(n//2,P)
 clockadd(Q,Q)
n % 2: Q = clockadd(P,Q)
urn Q


ursixdigitsecret
rmult(n,P)
736284)
```

gure out our secret n?

Clock cryptography

The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.

Alice's se

Alice's p
$a($

{Alice
share
$ab($

```
,P):
urn (Fp(0),Fp(1))
urn P
(n//2,P)
,Q)
clockadd(P,Q)



ecret
```

secret n?

Clock cryptography

The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.

Alice's secret key a

Alice's public key
$a(x, y)$

{Alice, Bob}'s
shared secret
$ab(x, y)$

=

```
Fp(1))
```

```
Q)
```

Clock cryptography

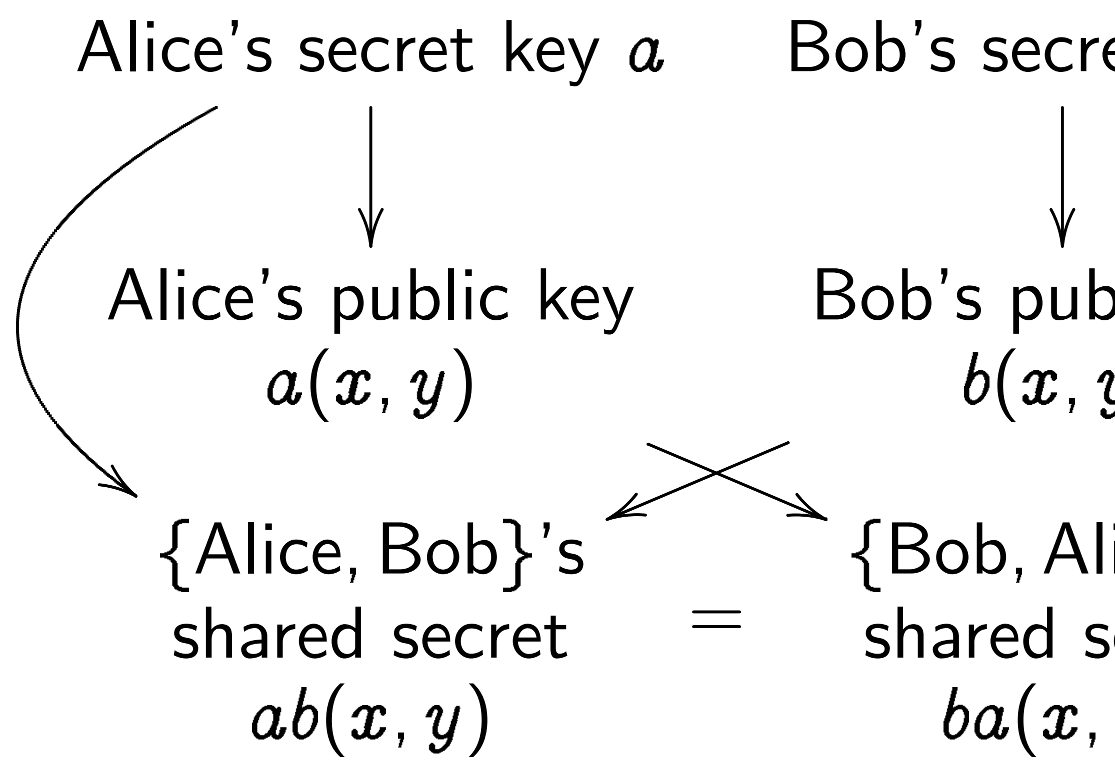
The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.



<u>Clock cryptography</u>

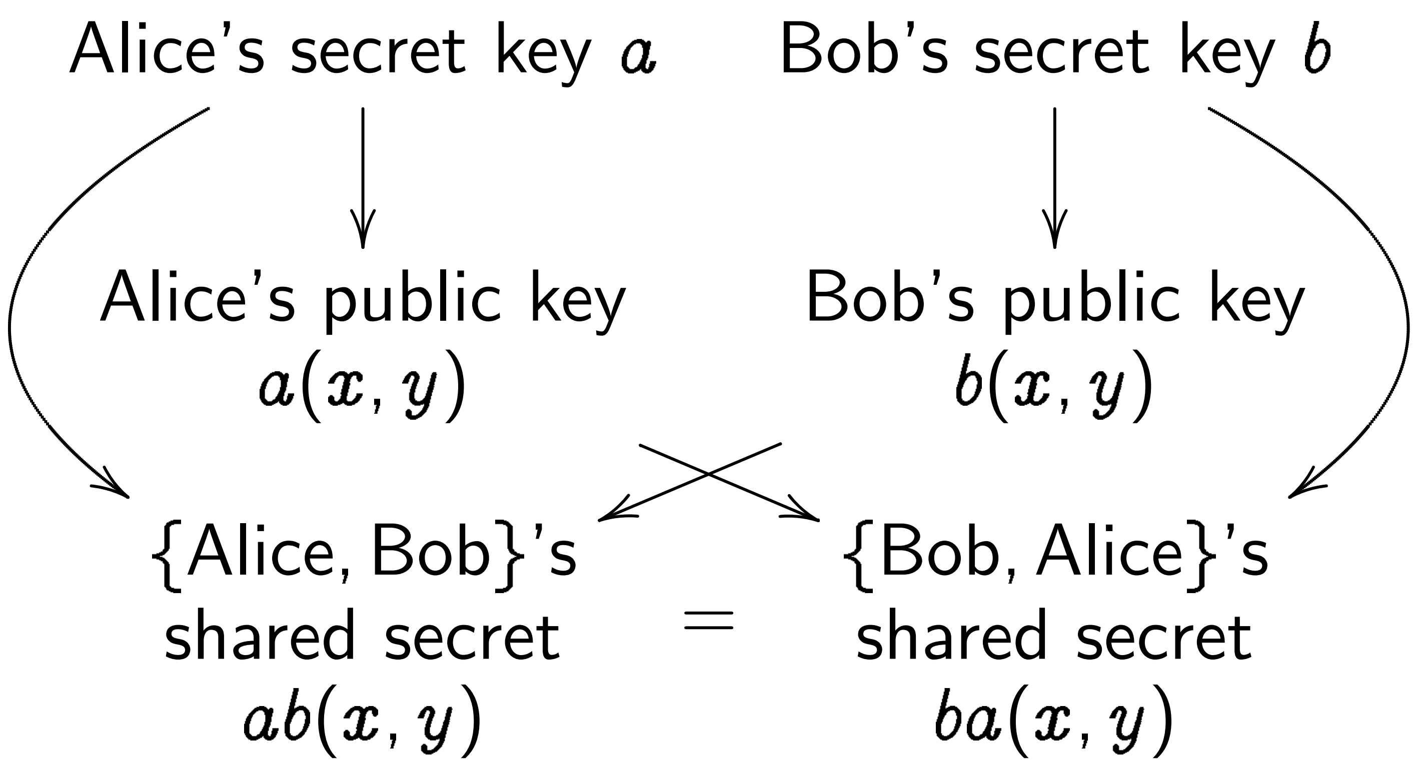
The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.



Clock cryptography

The "Clock Diffie–Hellman protocol":

Standardize a large prime p
and **base point** $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

Alice chooses big secret a.
Alice computes her public key $a(x, y)$.

Bob chooses big secret b.
Bob computes his public key $b(x, y)$.

Alice computes $a(b(x, y))$.
Bob computes $b(a(x, y))$.
They use this shared secret
to encrypt with AES-GCM etc.

Alice's secret key a | Bob's secret key b

Alice's public key $a(x, y)$ | Bob's public key $b(x, y)$

{Alice, Bob}'s shared secret $ab(x, y)$ $=$ {Bob, Alice}'s shared secret $ba(x, y)$

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!
Can use index calculus
to attack clock cryptography.
To match RSA-3072 security
need $p \approx 2^{1536}$.

tography

k Diffie–Hellman protocol":

e a large prime p

oint $(x, y) \in \text{Clock}(\mathbf{F}_p)$.

ses big secret a.

putes her public key $a(x, y)$.

es big secret b.

utes his public key $b(x, y)$.

putes $a(b(x, y))$.

utes $b(a(x, y))$.

his shared secret

with AES-GCM etc.

$$
\begin{tikzcd}
\text{Alice's secret key } a \arrow[d] \arrow[dd, bend right] & & \text{Bob's secret key } b \arrow[d] \arrow[dd, bend left] \\
\begin{array}{c}\text{Alice's public key}\\ a(x,y)\end{array} \arrow[drr] & & \begin{array}{c}\text{Bob's public key}\\ b(x,y)\end{array} \arrow[dll] \\
\begin{array}{c}\{\text{Alice, Bob}\}\text{'s}\\ \text{shared secret}\\ ab(x,y)\end{array} & = & \begin{array}{c}\{\text{Bob, Alice}\}\text{'s}\\ \text{shared secret}\\ ba(x,y)\end{array}
\end{tikzcd}
$$

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!

Can use index calculus

to attack clock cryptography.

To match RSA-3072 security

need $p \approx 2^{1536}$.

Warning #

the public

Attacker se

Alice uses

Often attac

for *each op*

not just to

This reveal

Some timin

2013 "Luck

2014 Beng

man protocol":

me p

$\in$ Clock($\mathbf{F}_p$).

t a.

blic key $a(x, y)$.

b.

ic key $b(x, y)$.

y)).

)).

cret

CM etc.

Alice's secret key a Bob's secret key b

Alice's public key $a(x, y)$

Bob's public key $b(x, y)$

{Alice, Bob}'s shared secret $ab(x, y)$ = {Bob, Alice}'s shared secret $ba(x, y)$

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!
Can use index calculus
to attack clock cryptography.
To match RSA-3072 security
need $p \approx 2^{1536}$.

Warning #3: Attacker
the public keys $a(x, y)$

Attacker sees how much
Alice uses to compute
Often attacker can see
for *each operation* perf
not just total time.
This reveals secret scal

Some timing attacks: 2
2013 "Lucky Thirteen"
2014 Benger–van de Po

l":

$y)$.

).

Alice's secret key a Bob's secret key b

Alice's public key $a(x, y)$ Bob's public key $b(x, y)$

{Alice, Bob}'s shared secret $ab(x, y)$ = {Bob, Alice}'s shared secret $ba(x, y)$

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!
Can use index calculus
to attack clock cryptography.
To match RSA-3072 security
need $p \approx 2^{1536}$.

Warning #3: Attacker sees more th
the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time*
Alice uses to compute $a(b(x, y))$.
Often attacker can see time
for *each operation* performed by Ali
not just total time.
This reveals secret scalar a.

Some timing attacks: 2011 Brumley
2013 "Lucky Thirteen" (not ECC);
2014 Benger–van de Pol–Smart–Yar

Alice's secret key a Bob's secret key b

Alice's public key $a(x, y)$ Bob's public key $b(x, y)$

{Alice, Bob}'s shared secret $ab(x, y)$ $=$ {Bob, Alice}'s shared secret $ba(x, y)$

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!
Can use index calculus
to attack clock cryptography.
To match RSA-3072 security
need $p \approx 2^{1536}$.

Warning #3: Attacker sees more than
the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time*
Alice uses to compute $a(b(x, y))$.
Often attacker can see time
for *each operation* performed by Alice,
not just total time.
This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri;
2013 "Lucky Thirteen" (not ECC);
2014 Benger–van de Pol–Smart–Yarom; etc.

$$
\begin{tikzcd}
\text{Alice's secret key } a \arrow[d] \arrow[dd, bend right] & & \text{Bob's secret key } b \arrow[d] \arrow[dd, bend left] \\
\begin{array}{c}\text{Alice's public key} \\ a(x,y)\end{array} \arrow[drr] & & \begin{array}{c}\text{Bob's public key} \\ b(x,y)\end{array} \arrow[dll] \\
\begin{array}{c}\{\text{Alice, Bob}\}\text{'s} \\ \text{shared secret} \\ ab(x,y)\end{array} & = & \begin{array}{c}\{\text{Bob, Alice}\}\text{'s} \\ \text{shared secret} \\ ba(x,y)\end{array}
\end{tikzcd}
$$

Warning #1: Many choices of p are unsafe!

Warning #2: Clocks aren't elliptic!
Can use index calculus
to attack clock cryptography.
To match RSA-3072 security
need $p \approx 2^{1536}$.

Warning #3: Attacker sees more than
the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time*
Alice uses to compute $a(b(x, y))$.
Often attacker can see time
for *each operation* performed by Alice,
not just total time.
This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri;
2013 "Lucky Thirteen" (not ECC);
2014 Benger–van de Pol–Smart–Yarom; etc.

Fix: **constant-time** code,
performing same operations
no matter what scalar is.

ecret key a Bob's secret key b

public key Bob's public key

$x, y)$ $b(x, y)$

, Bob}'s {Bob, Alice}'s

d secret $=$ shared secret

$x, y)$ $ba(x, y)$

1: Many choices of p are unsafe!

2: Clocks aren't elliptic!

dex calculus

clock cryptography.

RSA-3072 security

^{1536}.

Warning #3: Attacker sees more than the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time* Alice uses to compute $a(b(x, y))$. Often attacker can see time for *each operation* performed by Alice, not just total time. This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri; 2013 "Lucky Thirteen" (not ECC); 2014 Benger–van de Pol–Smart–Yarom; etc.

Fix: **constant-time** code, performing same operations no matter what scalar is.

Addition o

$x^2 + y^2 =$

Sum of $(x_1$

$((x_1y_2+y_1$

$(y_1y_2-x_1$

Bob's secret key b

Bob's public key
$b(x, y)$

{Bob, Alice}'s
shared secret
$ba(x, y)$

oices of p are unsafe!

ren't elliptic!

raphy.

curity

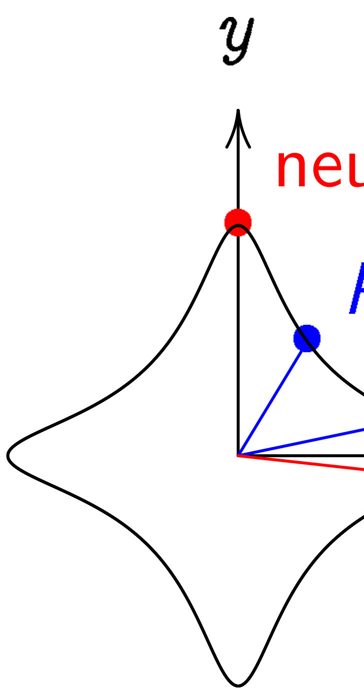
Warning #3: Attacker sees more than the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time* Alice uses to compute $a(b(x, y))$. Often attacker can see time for *each operation* performed by Alice, not just total time. This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri; 2013 "Lucky Thirteen" (not ECC); 2014 Benger–van de Pol–Smart–Yarom; etc.

Fix: **constant-time** code, performing same operations no matter what scalar is.

Addition on an elliptic c



$x^2 + y^2 = 1 - 30x^2y^2$.
Sum of (x_1, y_1) and $(x$
$((x_1y_2{+}y_1x_2)/(1{-}30x$
$(y_1y_2{-}x_1x_2)/(1{+}30x$

et key b

lic key
$y)$

ice}'s
ecret
$y)$

e unsafe!

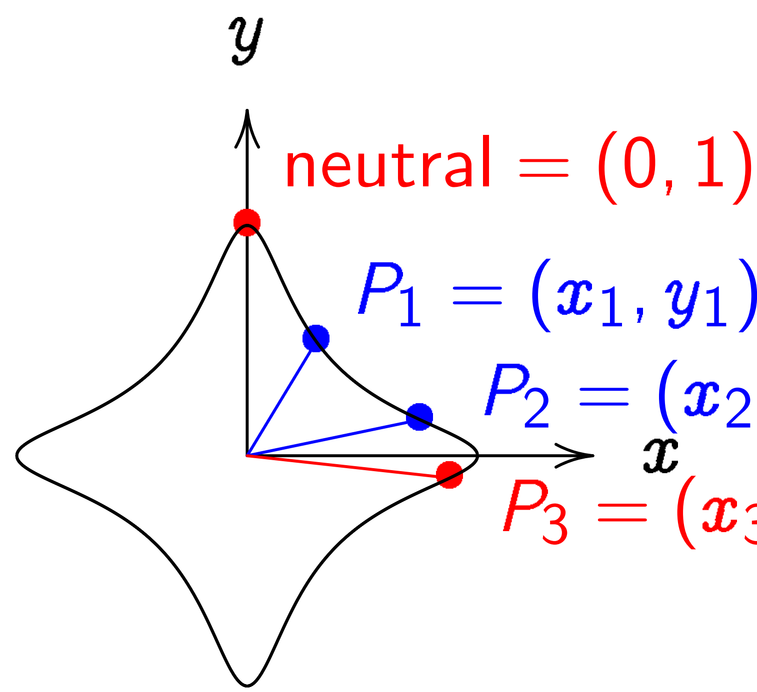
Warning #3: Attacker sees more than the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time* Alice uses to compute $a(b(x, y))$. Often attacker can see time for *each operation* performed by Alice, not just total time. This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri; 2013 "Lucky Thirteen" (not ECC); 2014 Benger–van de Pol–Smart–Yarom; etc.

Fix: **constant-time** code, performing same operations no matter what scalar is.

Addition on an elliptic curve



$x^2 + y^2 = 1 - 30x^2y^2$.
Sum of (x_1, y_1) and (x_2, y_2) is
$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

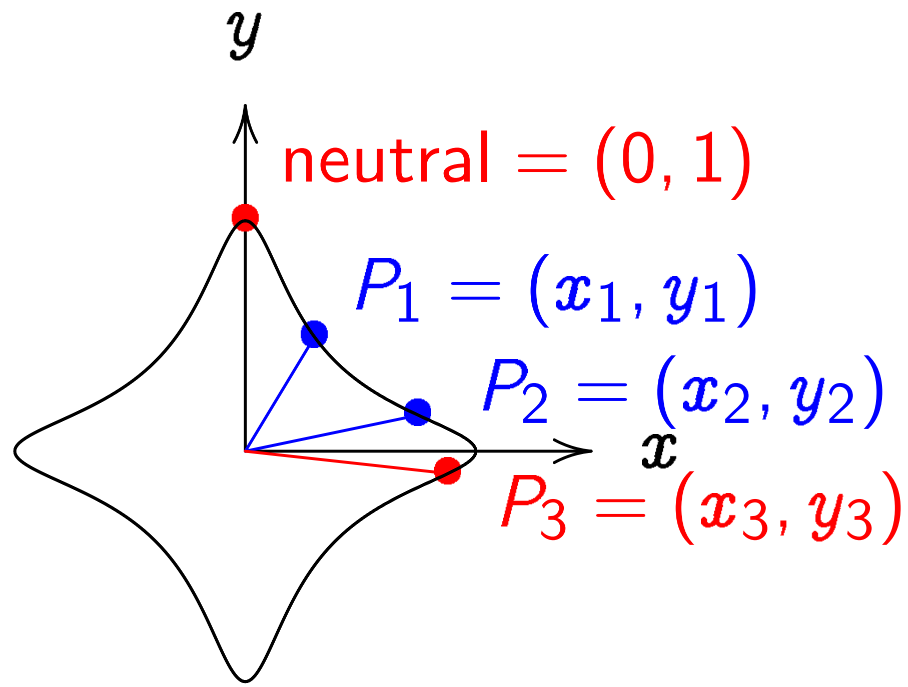
Warning #3: Attacker sees more than the public keys $a(x, y)$ and $b(x, y)$.

Attacker sees how much *time* Alice uses to compute $a(b(x, y))$. Often attacker can see time for *each operation* performed by Alice, not just total time. This reveals secret scalar a.

Some timing attacks: 2011 Brumley–Tuveri; 2013 "Lucky Thirteen" (not ECC); 2014 Benger–van de Pol–Smart–Yarom; etc.

Fix: **constant-time** code, performing same operations no matter what scalar is.

Addition on an elliptic curve



$x^2 + y^2 = 1 - 30x^2y^2$.

Sum of (x_1, y_1) and (x_2, y_2) is

$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$

$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

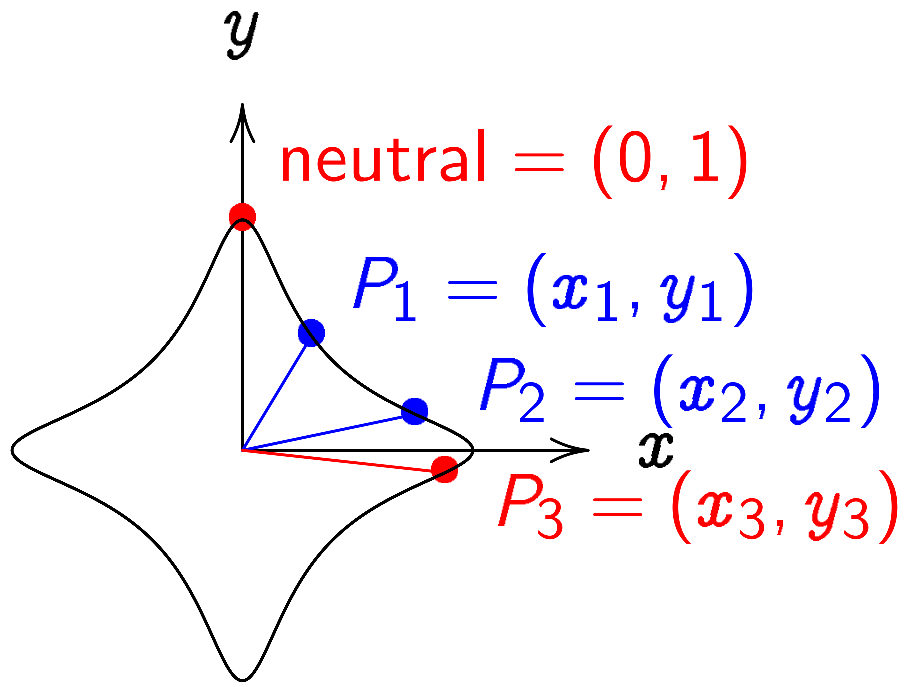
3: Attacker sees more than
keys $a(x, y)$ and $b(x, y)$.

ees how much *time*
to compute $a(b(x, y))$.
cker can see time
peration performed by Alice,
tal time.
s secret scalar a.

ng attacks: 2011 Brumley–Tuveri;
ky Thirteen" (not ECC);
er–van de Pol–Smart–Yarom; etc.

ant-time code,
same operations
what scalar is.

Addition on an elliptic curve



$x^2 + y^2 = 1 - 30x^2y^2$.
Sum of (x_1, y_1) and (x_2, y_2) is
$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

The clock

$x^2 + y^2 =$
Sum of $(x_1$
$(x_1y_2 + y_1$
$y_1y_2 - x_1$

sees more than
and $b(x, y)$.

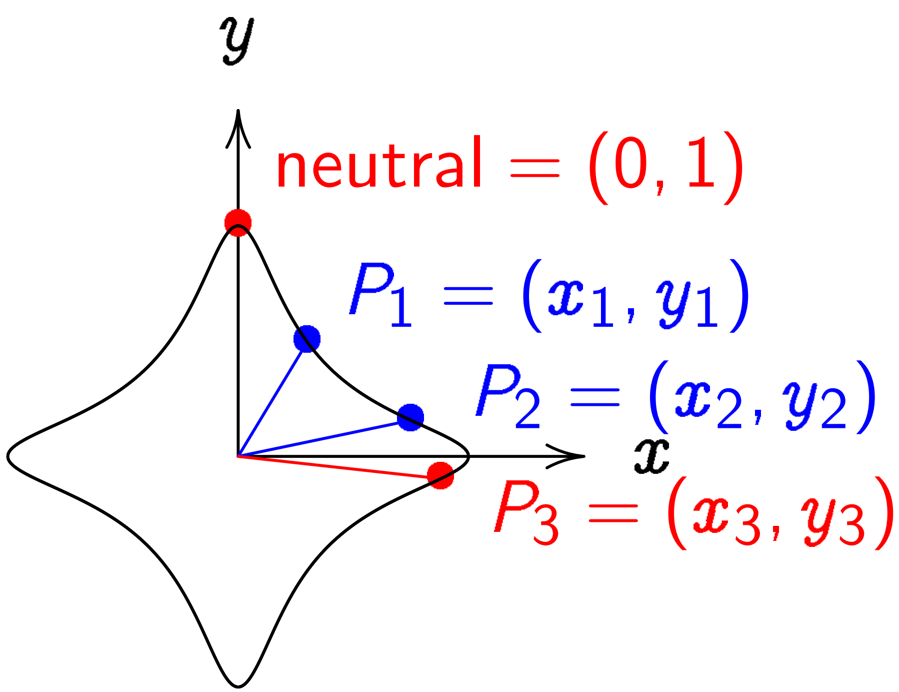
h *time*
$a(b(x, y))$.
time
ormed by Alice,

ar a.

2011 Brumley–Tuveri;
(not ECC);
ol–Smart–Yarom; etc.

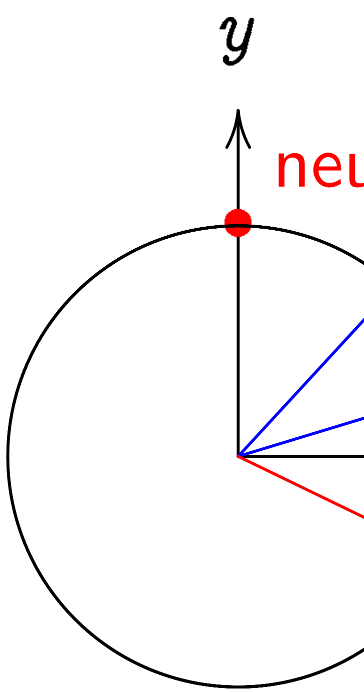
de,
tions
s.

Addition on an elliptic curve



$x^2 + y^2 = 1 - 30x^2y^2$.
Sum of (x_1, y_1) and (x_2, y_2) is
$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

The clock again, for co



$x^2 + y^2 = 1$.
Sum of (x_1, y_1) and $(x$
$(x_1y_2 + y_1x_2,$
$y_1y_2 - x_1x_2)$.

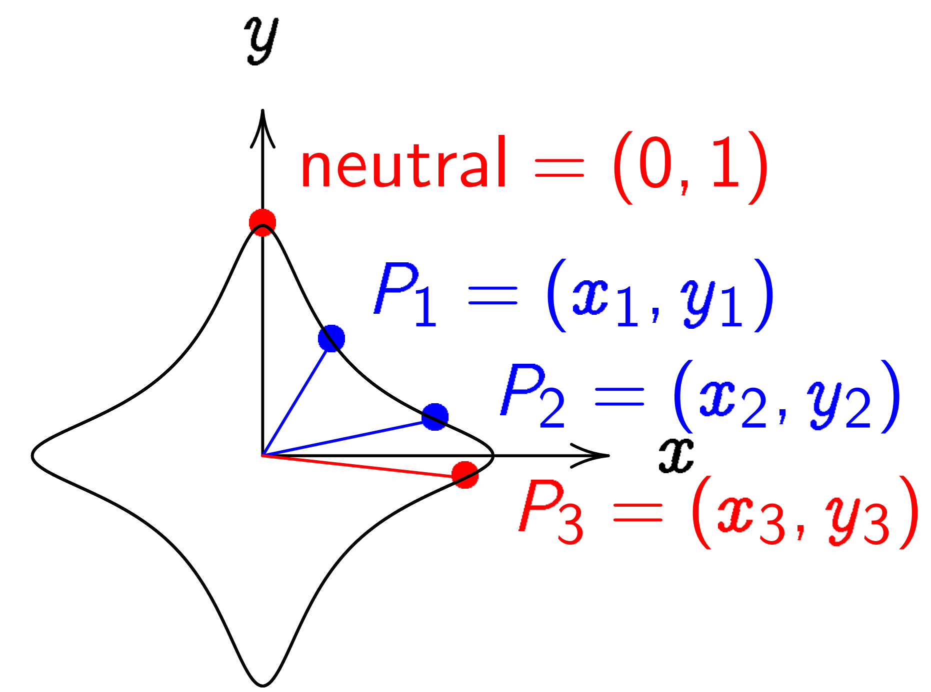
an

ce,

y–Tuveri;

rom; etc.

Addition on an elliptic curve



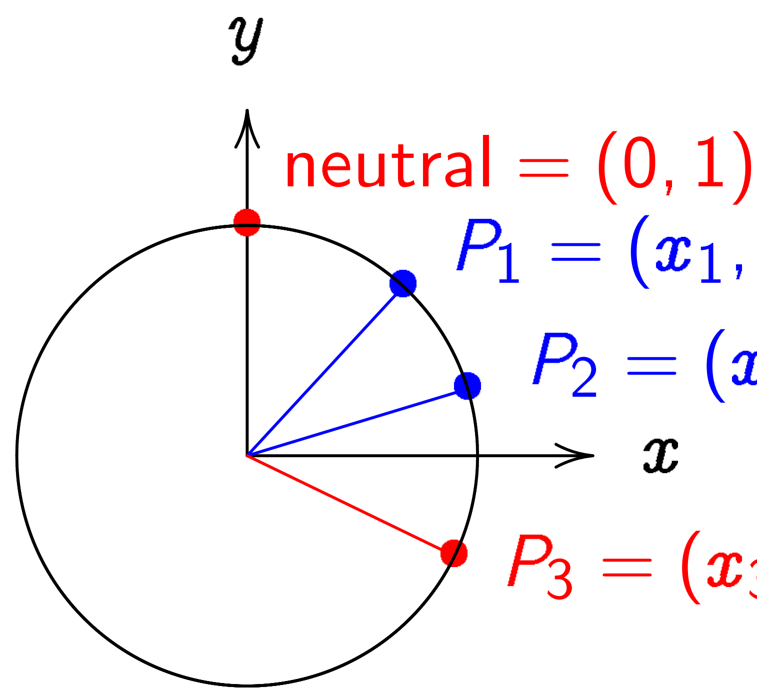
$x^2 + y^2 = 1 - 30x^2y^2$.

Sum of (x_1, y_1) and (x_2, y_2) is

$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$

$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

The clock again, for comparison:



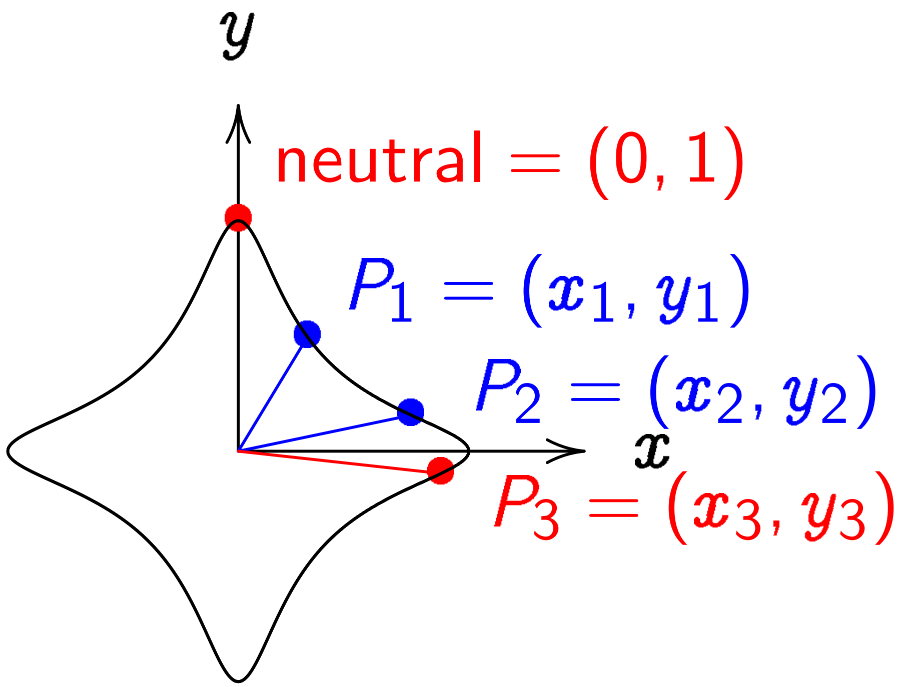
$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

Addition on an elliptic curve

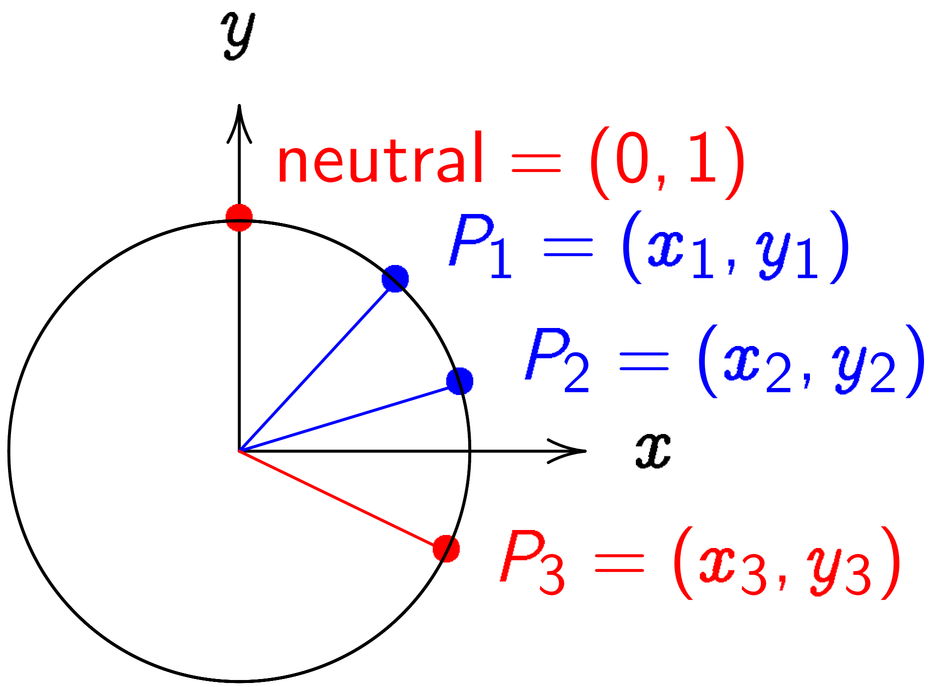


$x^2 + y^2 = 1 - 30x^2y^2$.

Sum of (x_1, y_1) and (x_2, y_2) is

$((x_1y_2+y_1x_2)/(1-30x_1x_2y_1y_2),$
$(y_1y_2-x_1x_2)/(1+30x_1x_2y_1y_2))$.

The clock again, for comparison:

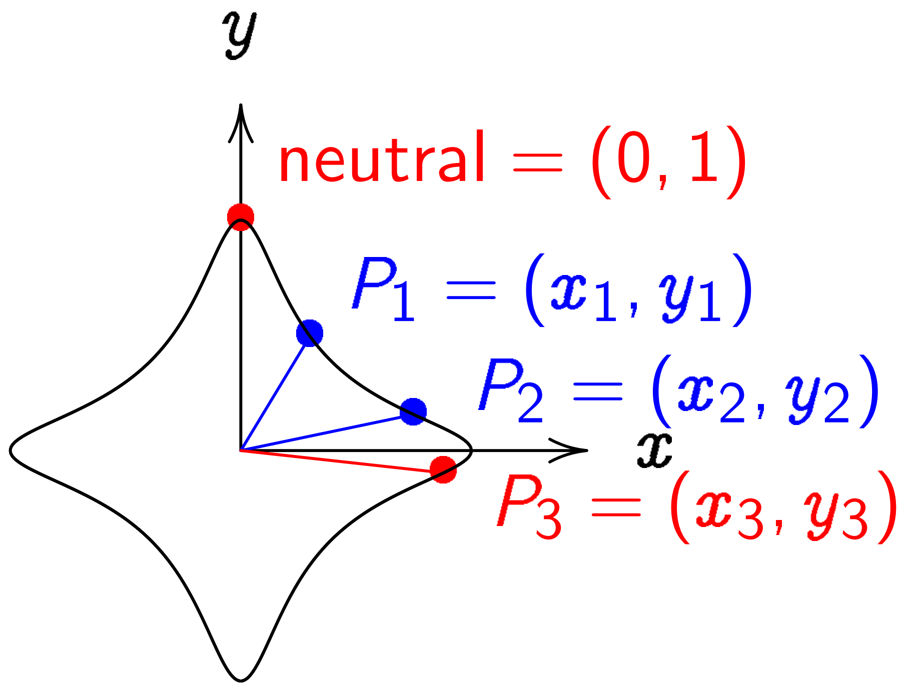


$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2,$
$y_1y_2 - x_1x_2)$.

n an elliptic curve



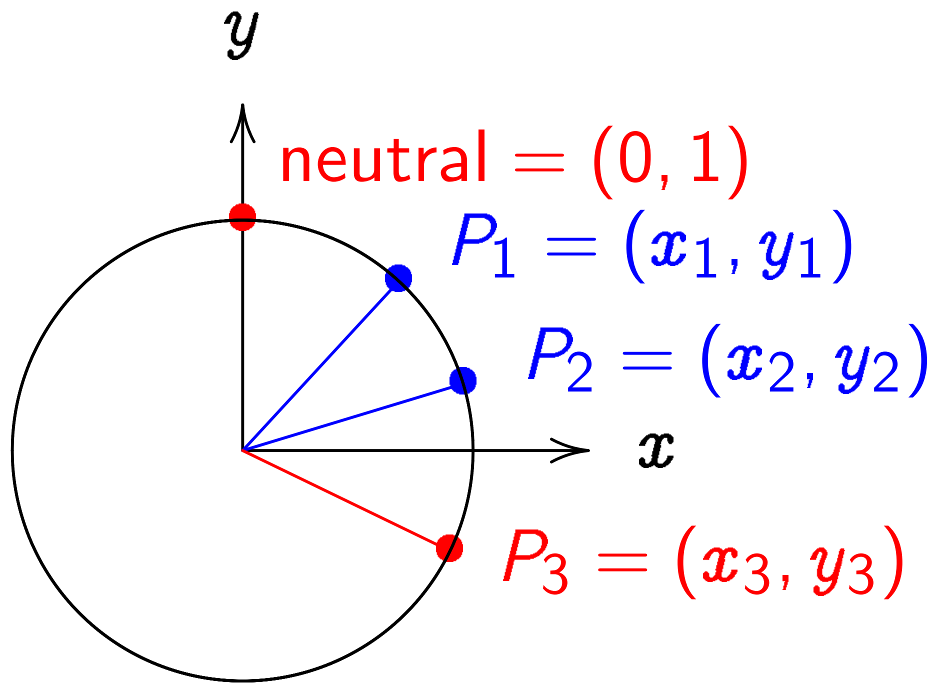
$1 - 30x^2y^2$.

$_1, y_1)$ and (x_2, y_2) is

$x_2)/(1{-}30x_1x_2y_1y_2)$,

$x_2)/(1{+}30x_1x_2y_1y_2))$.

The clock again, for comparison:



$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1y_2 + y_1x_2,$

$y_1y_2 - x_1x_2)$.

More ellipt

Choose an

Choose a *r*

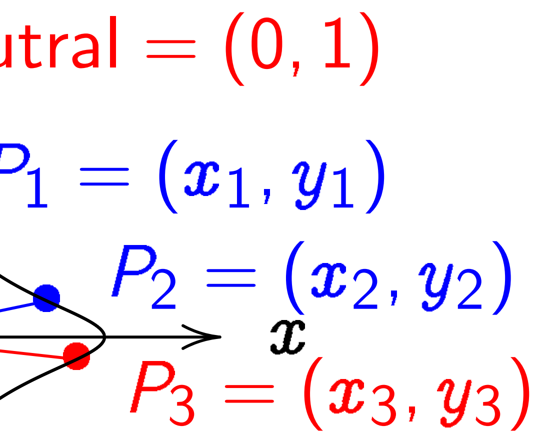
$\{(x, y) \in \mathbf{F}$

$x^2 + y$

is a "comp

```
def edwar
  x1,y1 =
  x2,y2 =
  x3 = (x1
  y3 = (y1
  return x
```

curve

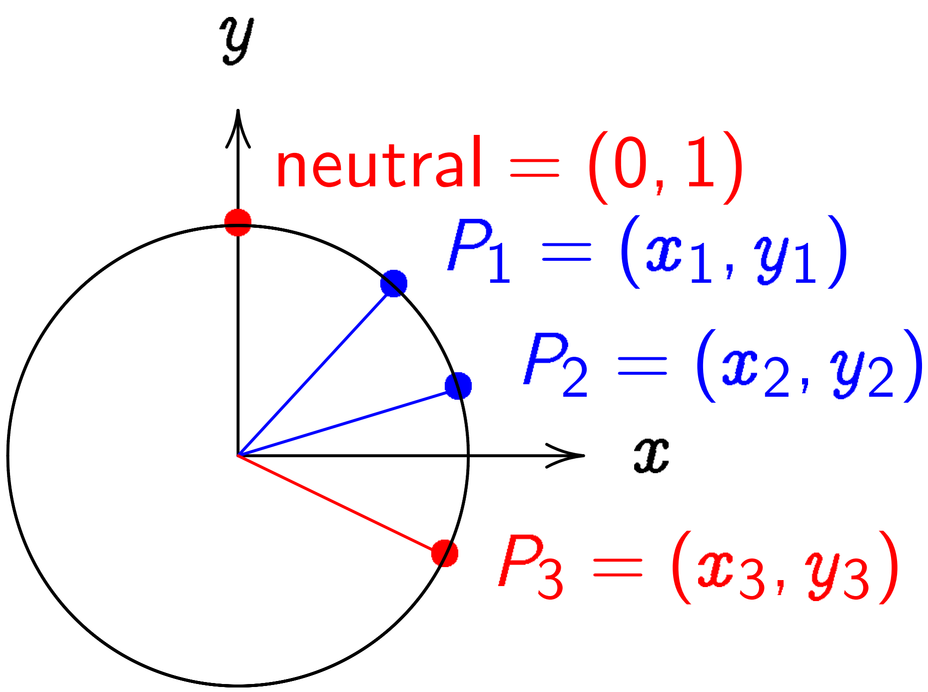


$_2, y_2)$ is

$_1 x_2 y_1 y_2)$,

$_1 x_2 y_1 y_2))$.

The clock again, for comparison:



$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2,$

$y_1 y_2 - x_1 x_2)$.

More elliptic curves

Choose an odd prime p

Choose a *non-square* d

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$

$x^2 + y^2 = 1 + dx^2$

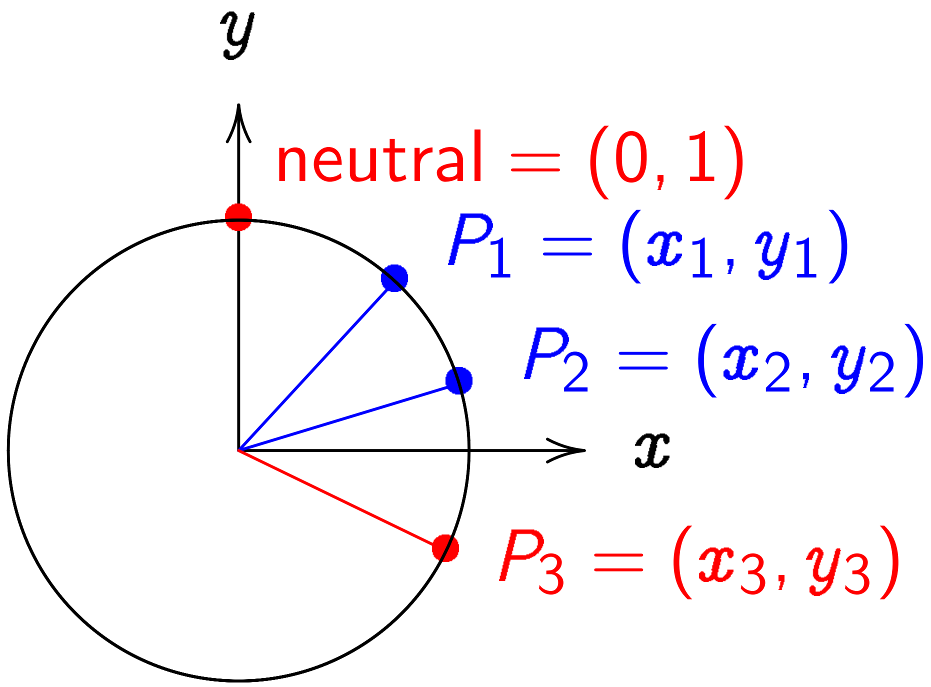
is a "complete Edwards

```
def edwardsadd(P1,P2)
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/
  y3 = (y1*y2-x1*x2)/
  return x3,y3
```

$, y_2)$

$_3, y_3)$

The clock again, for comparison:



$x^2 + y^2 = 1.$

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2,$

$y_1 y_2 - x_1 x_2).$

<u>More elliptic curves</u>

Choose an odd prime p.

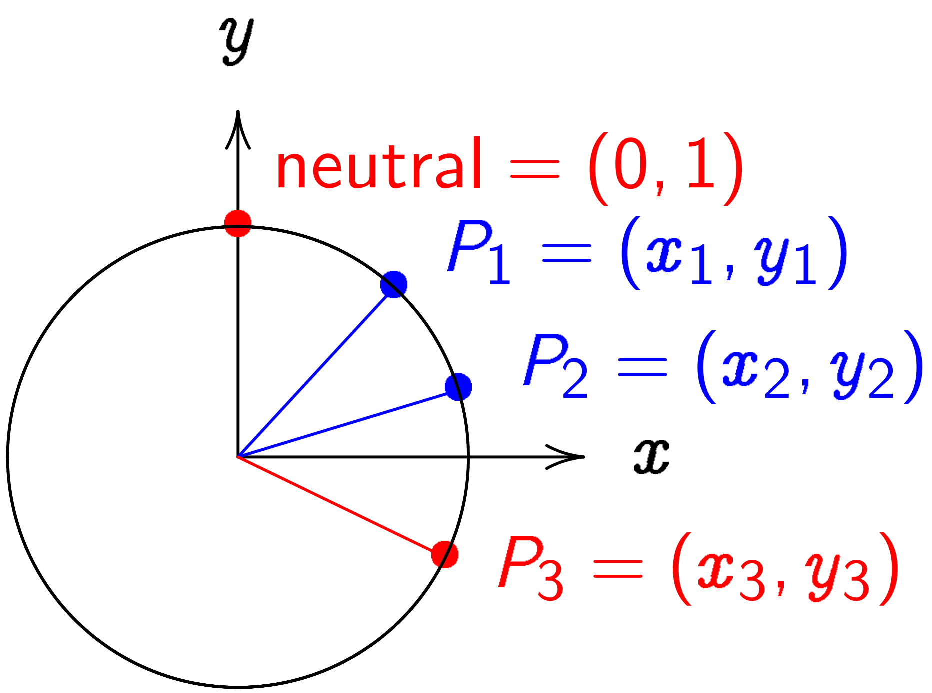
Choose a *non-square* $d \in \mathbf{F}_p$.

$$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$$
$$x^2 + y^2 = 1 + dx^2y^2\}$$

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*
  return x3,y3
```

The clock again, for comparison:



$x^2 + y^2 = 1$.

Sum of (x_1, y_1) and (x_2, y_2) is

$(x_1 y_2 + y_1 x_2,$

$y_1 y_2 - x_1 x_2)$.

More elliptic curves

Choose an odd prime p.

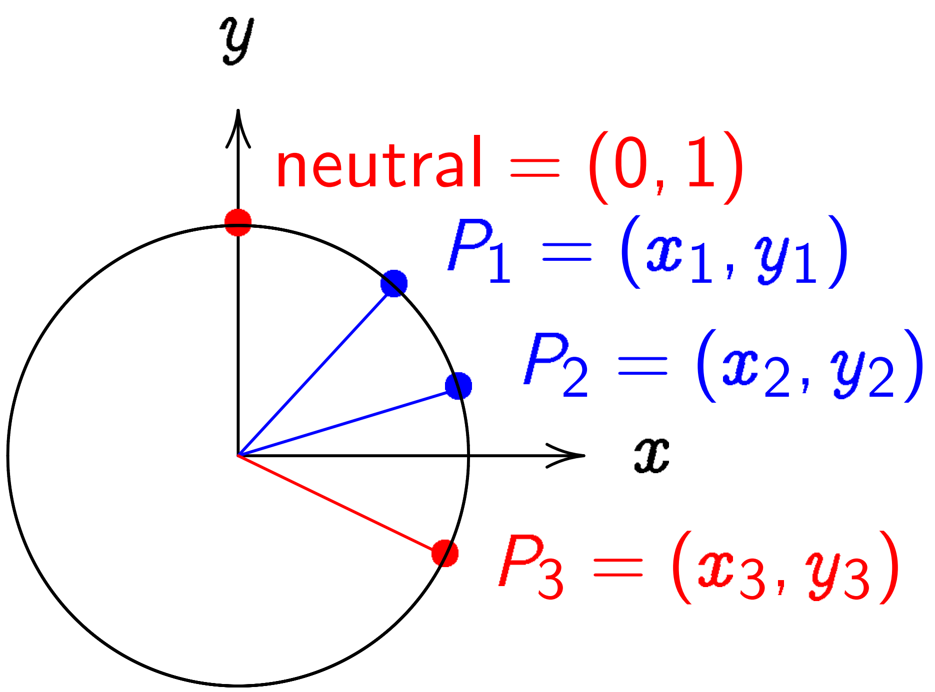
Choose a *non-square* $d \in \mathbf{F}_p$.

$$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p : x^2 + y^2 = 1 + dx^2y^2\}$$

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

again, for comparison:



1.

$_1, y_1)$ and (x_2, y_2) is

$x_2,$

$x_2).$

More elliptic curves

Choose an odd prime p.

Choose a *non-square* $d \in \mathbf{F}_p$.

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$
$\quad x^2 + y^2 = 1 + dx^2y^2\}$

is a "complete Edwards curve".

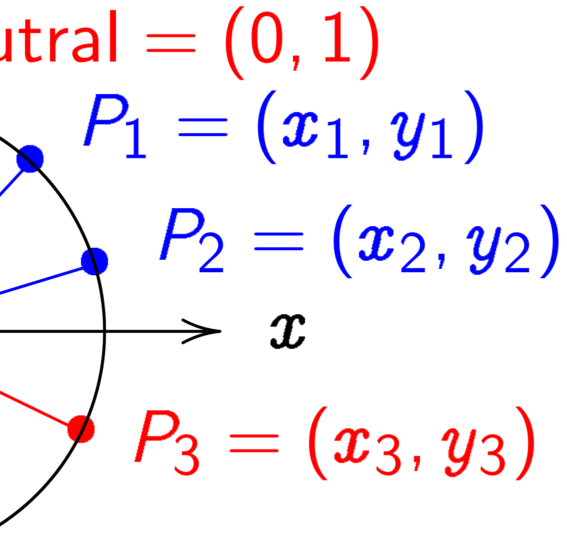
```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there

in the Edw

What if th

mparison:



$_2, y_2)$ is

More elliptic curves

Choose an odd prime p.

Choose a *non-square* $d \in \mathbf{F}_p$.

$$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p : \\ x^2 + y^2 = 1 + dx^2y^2\}$$

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there are division

in the Edwards additio

What if the denominat

$y_1)$

$_2, y_2)$

$_3, y_3)$

More elliptic curves

Choose an odd prime p.

Choose a *non-square* $d \in \mathbf{F}_p$.

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$

$$x^2 + y^2 = 1 + dx^2y^2\}$$

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there are divisions in the Edwards addition law! What if the denominators are 0?"

More elliptic curves

Choose an odd prime p.
Choose a *non-square* $d \in \mathbf{F}_p$.

$$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p : \\ x^2 + y^2 = 1 + dx^2y^2\}$$

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

More elliptic curves

Choose an odd prime p.
Choose a *non-square* $d \in \mathbf{F}_p$.

$$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p : x^2 + y^2 = 1 + dx^2y^2\}$$

is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

More elliptic curves

Choose an odd prime p.
Choose a *non-square* $d \in \mathbf{F}_p$.

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$
$$x^2 + y^2 = 1 + dx^2y^2\}$$
is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square* d.

More elliptic curves

Choose an odd prime p.
Choose a *non-square* $d \in \mathbf{F}_p$.

$\{(x, y) \in \mathbf{F}_p \times \mathbf{F}_p :$
$\quad x^2 + y^2 = 1 + dx^2y^2\}$
is a "complete Edwards curve".

```
def edwardsadd(P1,P2):
  x1,y1 = P1
  x2,y2 = P2
  x3 = (x1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
  y3 = (y1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
  return x3,y3
```

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square* d.

If we instead choose square d:
curve is still elliptic, and
addition *seems to work,*
but there are failure cases,
often exploitable by attackers.
Safe code is more complicated.

ic curves

odd prime p.

non-square $d \in \mathbf{F}_p$.

$\mathbf{F}_p \times \mathbf{F}_p$:

$y^2 = 1 + dx^2y^2\}$

lete Edwards curve".

```
dsadd(P1,P2):
 P1
 P2
1*y2+y1*x2)/(1+d*x1*x2*y1*y2)
1*y2-x1*x2)/(1-d*x1*x2*y1*y2)
x3,y3
```

"Hey, there are divisions in the Edwards addition law! What if the denominators are 0?"

Answer: Can prove that the denominators are never 0. Addition law is **complete**.

This proof relies on choosing *non-square* d.

If we instead choose square d: curve is still elliptic, and addition *seems to work,* but there are failure cases, often exploitable by attackers. Safe code is more complicated.

"Hey, divis

.

$\in \mathbf{F}_p$.

$y^2\}$

s curve".

) :

```
/(1+d*x1*x2*y1*y2)
/(1-d*x1*x2*y1*y2)
```

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square* d.

If we instead choose square d:
curve is still elliptic, and
addition *seems to work,*
but there are failure cases,
often exploitable by attackers.
Safe code is more complicated.

"Hey, divisions are reall

y1*y2)

y1*y2)

"Hey, there are divisions in the Edwards addition law! What if the denominators are 0?"

Answer: Can prove that the denominators are never 0. Addition law is **complete**.

This proof relies on choosing *non-square* d.

If we instead choose square d: curve is still elliptic, and addition *seems to work,* but there are failure cases, often exploitable by attackers. Safe code is more complicated.

"Hey, divisions are really slow!"

"Hey, there are divisions in the Edwards addition law! What if the denominators are 0?"

Answer: Can prove that the denominators are never 0. Addition law is **complete**.

This proof relies on choosing *non-square* d.

If we instead choose square d: curve is still elliptic, and addition *seems to work,* but there are failure cases, often exploitable by attackers. Safe code is more complicated.

"Hey, divisions are really slow!"

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square* d.

If we instead choose square d:
curve is still elliptic, and
addition *seems to work,*
but there are failure cases,
often exploitable by attackers.
Safe code is more complicated.

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

"Hey, there are divisions
in the Edwards addition law!
What if the denominators are 0?"

Answer: Can prove that
the denominators are never 0.
Addition law is **complete**.

This proof relies on
choosing *non-square* d.

If we instead choose square d:
curve is still elliptic, and
addition *seems to work,*
but there are failure cases,
often exploitable by attackers.
Safe code is more complicated.

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

One option: "projective coordinates".
Store (X, Y, Z) representing $(X/Z, Y/Z)$.

Another option: "extended coordinates".
Store projective (X, Y, Z) and $T = XY/Z$.

See "Explicit Formulas Database"
for many more options and speedups:
`hyperelliptic.org/EFD`

e are divisions
ards addition law!
e denominators are 0?"

an prove that
inators are never 0.
w is **complete**.

relies on
on-square d.

ad choose square d:
ll elliptic, and
eems to work,
are failure cases,
oitable by attackers.
is more complicated.

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

One option: "projective coordinates".
Store (X, Y, Z) representing $(X/Z, Y/Z)$.

Another option: "extended coordinates".
Store projective (X, Y, Z) and $T = XY/Z$.

See "Explicit Formulas Database"
for many more options and speedups:
hyperelliptic.org/EFD

Elliptic-cur

Standardiz
base point

Alice know
and Bob's
Alice comp
shared secr

Alice uses
and auther

Packet ove
32 bytes fo
24 bytes fo
16 bytes fo

s

n law!

ors are 0?"

t

ever 0.

te.

uare d:

d

,

ses,

ackers.

plicated.

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

One option: "projective coordinates".
Store (X, Y, Z) representing $(X/Z, Y/Z)$.

Another option: "extended coordinates".
Store projective (X, Y, Z) and $T = XY/Z$.

See "Explicit Formulas Database"
for many more options and speedups:
hyperelliptic.org/EFD

Elliptic-curve cryptogra

Standardize prime p, sa
base point (x, y) on elli

Alice knows her secret
and Bob's public key $b($
Alice computes (and ca
shared secret $ab(x, y)$.

Alice uses shared secret
and authenticate packe

Packet overhead at hig
32 bytes for Alice's pub
24 bytes for nonce,
16 bytes for authentica

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

One option: "projective coordinates".
Store (X, Y, Z) representing $(X/Z, Y/Z)$.

Another option: "extended coordinates".
Store projective (X, Y, Z) and $T = XY/Z$.

See "Explicit Formulas Database"
for many more options and speedups:
hyperelliptic.org/EFD

Elliptic-curve cryptography

Standardize prime p, safe non-squar
base point (x, y) on elliptic curve.

Alice knows her secret key a
and Bob's public key $b(x, y)$.
Alice computes (and caches)
shared secret $ab(x, y)$.

Alice uses shared secret to encrypt
and authenticate packet for Bob.

Packet overhead at high security lev
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

"Hey, divisions are really slow!"

Instead of dividing a by b,
store fraction a/b as pair (a, b).
Remember arithmetic on fractions?

One option: "projective coordinates".
Store (X, Y, Z) representing $(X/Z, Y/Z)$.

Another option: "extended coordinates".
Store projective (X, Y, Z) and $T = XY/Z$.

See "Explicit Formulas Database"
for many more options and speedups:
hyperelliptic.org/EFD

Elliptic-curve cryptography

Standardize prime p, safe non-square d,
base point (x, y) on elliptic curve.

Alice knows her secret key a
and Bob's public key $b(x, y)$.
Alice computes (and caches)
shared secret $ab(x, y)$.

Alice uses shared secret to encrypt
and authenticate packet for Bob.

Packet overhead at high security level:
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

ions are really slow!"

dividing a by b,
on a/b as pair (a, b).
arithmetic on fractions?

n: "projective coordinates".
$Y, Z)$ representing $(X/Z, Y/Z)$.

ption: "extended coordinates".
ective (X, Y, Z) and $T = XY/Z$.

cit Formulas Database"
more options and speedups:
iptic.org/EFD

Elliptic-curve cryptography

Standardize prime p, safe non-square d,
base point (x, y) on elliptic curve.

Alice knows her secret key a
and Bob's public key $b(x, y)$.
Alice computes (and caches)
shared secret $ab(x, y)$.

Alice uses shared secret to encrypt
and authenticate packet for Bob.

Packet overhead at high security level:
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

Bob receiv
sees Alice's
Bob compu
shared secr

Bob uses s
verify auth

Alice and B
reuse the s
encrypt, au
all subsequ

All of this
we can affo

y slow!"

b,
ir (a, b).
on fractions?

e coordinates".
nting $(X/Z, Y/Z)$.

nded coordinates".
$Z)$ and $T = XY/Z$.

Database"
and speedups:
EFD

Elliptic-curve cryptography

Standardize prime p, safe non-square d, base point (x, y) on elliptic curve.

Alice knows her secret key a and Bob's public key $b(x, y)$. Alice computes (and caches) shared secret $ab(x, y)$.

Alice uses shared secret to encrypt and authenticate packet for Bob.

Packet overhead at high security level:
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

Bob receives packet,
sees Alice's public key a
Bob computes (and cac
shared secret $ab(x, y)$.

Bob uses shared secret
verify authenticator and

Alice and Bob
reuse the same shared s
encrypt, authenticate, v
all subsequent packets.

All of this is so fast tha
we can afford to encryp

s".

Y/Z).

ates".

XY/Z.

s:

Elliptic-curve cryptography

Standardize prime p, safe non-square d, base point (x, y) on elliptic curve.

Alice knows her secret key a and Bob's public key $b(x, y)$. Alice computes (and caches) shared secret $ab(x, y)$.

Alice uses shared secret to encrypt and authenticate packet for Bob.

Packet overhead at high security level:
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

Bob receives packet, sees Alice's public key $a(x, y)$. Bob computes (and caches) shared secret $ab(x, y)$.

Bob uses shared secret to verify authenticator and decrypt pac

Alice and Bob
reuse the same shared secret to encrypt, authenticate, verify, and de
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets

Elliptic-curve cryptography

Standardize prime p, safe non-square d, base point (x, y) on elliptic curve.

Alice knows her secret key a and Bob's public key $b(x, y)$. Alice computes (and caches) shared secret $ab(x, y)$.

Alice uses shared secret to encrypt and authenticate packet for Bob.

Packet overhead at high security level:
32 bytes for Alice's public key,
24 bytes for nonce,
16 bytes for authenticator.

Bob receives packet, sees Alice's public key $a(x, y)$. Bob computes (and caches) shared secret $ab(x, y)$.

Bob uses shared secret to verify authenticator and decrypt packet.

Alice and Bob reuse the same shared secret to encrypt, authenticate, verify, and decrypt all subsequent packets.

All of this is so fast that *we can afford to encrypt all packets.*

ve cryptography

e prime p, safe non-square d,
(x, y) on elliptic curve.

s her secret key a
public key $b(x, y)$.
utes (and caches)
ret $ab(x, y)$.

shared secret to encrypt
nticate packet for Bob.

rhead at high security level:
r Alice's public key,
r nonce,
r authenticator.

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe exa

Choose p =
Choose d =
this is non-

$x^2 + y^2 =$
is a safe cu

phy

afe non-square d,
iptic curve.

key a
(x, y).
aches)

t to encrypt
t for Bob.

h security level:
blic key,

tor.

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/12$
this is non-square in $\mathbf{F}_p$

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

e d,

vel:

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$
is another safe curve
using the same p and d.

Bob receives packet,
sees Alice's public key $a(x, y)$.
Bob computes (and caches)
shared secret $ab(x, y)$.

Bob uses shared secret to
verify authenticator and decrypt packet.

Alice and Bob
reuse the same shared secret to
encrypt, authenticate, verify, and decrypt
all subsequent packets.

All of this is so fast that
we can afford to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$
is another safe curve
using the same p and d.

Actually, the second curve
is the first curve in disguise:
replace x in first curve
by $\sqrt{-1} \cdot x$, using $\sqrt{-1} \in \mathbf{F}_p$.

es packet,
s public key $a(x, y)$.
utes (and caches)
ret $ab(x, y)$.

hared secret to
enticator and decrypt packet.

Bob
ame shared secret to
uthenticate, verify, and decrypt
ent packets.

is so fast that
ord to encrypt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$
is another safe curve
using the same p and d.

Actually, the second curve
is the first curve in disguise:
replace x in first curve
by $\sqrt{-1} \cdot x$, using $\sqrt{-1} \in \mathbf{F}_p$.

Even more

Edwards cu
$x^2 + y^2 =$

Twisted Ed
$ax^2 + y^2 =$

Weierstrass
$y^2 = x^3 +$

Montgome
$By^2 = x^3$

Many relat
e.g., obtain
given Mont
computing

$a(x, y)$.
ches)

to
d decrypt packet.

secret to
verify, and decrypt

nt
pt all packets.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$
is another safe curve
using the same p and d.

Actually, the second curve
is the first curve in disguise:
replace x in first curve
by $\sqrt{-1} \cdot x$, using $\sqrt{-1} \in \mathbf{F}_p$.

Even more elliptic curve

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards $(x$
given Montgomery $(x',$
computing $x = x'/y'$, y

cket.

ecrypt

r.

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$
is another safe curve
using the same p and d.

Actually, the second curve
is the first curve in disguise:
replace x in first curve
by $\sqrt{-1} \cdot x$, using $\sqrt{-1} \in \mathbf{F}_p$.

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/$

A safe example

Choose $p = 2^{255} - 19$.
Choose $d = 121665/121666$;
this is non-square in $\mathbf{F}_p$.

$x^2 + y^2 = 1 + dx^2y^2$
is a safe curve for ECC.

$-x^2 + y^2 = 1 - dx^2y^2$
is another safe curve
using the same p and d.

Actually, the second curve
is the first curve in disguise:
replace x in first curve
by $\sqrt{-1} \cdot x$, using $\sqrt{-1} \in \mathbf{F}_p$.

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

mple

$= 2^{255} - 19.$

$= 121665/121666;$

-square in $\mathbf{F}_p$.

$1 + dx^2y^2$

urve for ECC.

$= 1 - dx^2y^2$

safe curve

ame p and d.

ne second curve

curve in disguise:

n first curve

c, using $\sqrt{-1} \in \mathbf{F}_p$.

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition o

$y^2 = x^3 +$

1666;

.

r

y.

rve

guise:

$\in \mathbf{F}_p$.

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass
$y^2 = x^3 + a_4x + a_6$:

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass curves
$y^2 = x^3 + a_4x + a_6$:

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4x + a_6$:

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y)
given Montgomery (x', y') by
computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.

Even more elliptic curves

Edwards curves:
$x^2 + y^2 = 1 + dx^2y^2$.

Twisted Edwards curves:
$ax^2 + y^2 = 1 + dx^2y^2$.

Weierstrass curves:
$y^2 = x^3 + a_4x + a_6$.

Montgomery curves:
$By^2 = x^3 + Ax^2 + x$.

Many relationships:
e.g., obtain Edwards (x, y) given Montgomery (x', y') by computing $x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ with $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, $\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$ with $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda(x_1 - x_3) - y_1$, $\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

elliptic curves

urves:

$1 + dx^2y^2$.

dwards curves:

$= 1 + dx^2y^2$.

s curves:

$a_4x + a_6$.

ry curves:

$+ Ax^2 + x$.

ionships:

n Edwards (x, y)

tgomery (x', y') by

$x = x'/y'$, $y = (x' - 1)/(x' + 1)$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4x + a_6$:

for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;

for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;

$(x_1, y_1) + (x_1, -y_1) = \infty$;

$(x_1, y_1) + \infty = (x_1, y_1)$;

$\infty + (x_2, y_2) = (x_2, y_2)$;

$\infty + \infty = \infty$.

Messy to implement and test.

Much nice

curves with

```
def scala
  x2,z2,x
  for i i
    bit =
    x2,x3
    z2,z3
    x3,z3

    x2,z2


    x2,x3
    z2,z3
  return
```

es

s:

$, y)$

$y')$ by

$= (x'-1)/(x'+1)$.

Addition on Weierstrass curves

$y^2 = x^3 + a_4x + a_6$:

for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;

for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;

$(x_1, y_1) + (x_1, -y_1) = \infty$;

$(x_1, y_1) + \infty = (x_1, y_1)$;

$\infty + (x_2, y_2) = (x_2, y_2)$;

$\infty + \infty = \infty$.

Messy to implement and test.

Much nicer than Weiers
curves with the "Montg

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x
  for i in reversed(r
    bit = 1 & (n >> i
    x2,x3 = cswap(x2,
    z2,z3 = cswap(z2,
    x3,z3 = ((x2*x3-z
            x1*(x2*z3
    x2,z2 = ((x2^2-z2
            4*x2*z2*(
    x2,x3 = cswap(x2,
    z2,z3 = cswap(z2,
  return x2*z2^(p-2)
```

$'(x' + 1)$.

Addition on Weierstrass curves
$y^2 = x^3 + a_4x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

Much nicer than Weierstrass: Mont
curves with the "Montgomery ladde

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbi
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

Addition on Weierstrass curves

$y^2 = x^3 + a_4 x + a_6$:

for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;

for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;

$(x_1, y_1) + (x_1, -y_1) = \infty$;

$(x_1, y_1) + \infty = (x_1, y_1)$;

$\infty + (x_2, y_2) = (x_2, y_2)$;

$\infty + \infty = \infty$.

Messy to implement and test.

Much nicer than Weierstrass: Montgomery curves with the “Montgomery ladder”.

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbits)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

n Weierstrass curves

$a_4 x + a_6$:

$_2$, $(x_1, y_1) + (x_2, y_2) =$

th $x_3 = \lambda^2 - x_1 - x_2$,

$- x_3) - y_1$,

$y_1)/(x_2 - x_1)$;

$(x_1, y_1) + (x_1, y_1) =$

th $x_3 = \lambda^2 - x_1 - x_2$,

$- x_3) - y_1$,

$- a_4)/2y_1$;

$(x_1, -y_1) = \infty$;

$\infty = (x_1, y_1)$;

$_2) = (x_2, y_2)$;

∞.

mplement and test.

Much nicer than Weierstrass: Montgomery curves with the "Montgomery ladder".

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbits)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
                x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
                4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

Curve selec

How to def

an attacker

1999 ANSI

2000 IEEE

2000 Certic

2000 NIST

2001 ANSI

2005 Brain

2005 NSA

2010 Certic

2010 OSCC

2011 ANSS

s curves

$(x_2, y_2) =$

$- x_1 - x_2,$

$_1);$

$x_1, y_1) =$

$- x_1 - x_2,$

$\infty;$

);

);

d test.

Much nicer than Weierstrass: Montgomery curves with the "Montgomery ladder".

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbits)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

Curve selection

How to defend yourself
an attacker armed with

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v
2010 OSCCA SM2.
2011 ANSSI FRP256V1

Much nicer than Weierstrass: Montgomery curves with the "Montgomery ladder".

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbits)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

Curve selection

How to defend yourself against an attacker armed with a mathemat

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

Much nicer than Weierstrass: Montgomery curves with the “Montgomery ladder”.

```
def scalarmult(n,x1):
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(maxnbits)):
    bit = 1 & (n >> i)
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
    x3,z3 = ((x2*x3-z2*z3)^2,
             x1*(x2*z3-z2*x3)^2)
    x2,z2 = ((x2^2-z2^2)^2,
             4*x2*z2*(x2^2+A*x2*z2+z2^2))
    x2,x3 = cswap(x2,x3,bit)
    z2,z3 = cswap(z2,z3,bit)
  return x2*z2^(p-2)
```

Curve selection

How to defend yourself against an attacker armed with a mathematician:

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

r than Weierstrass: Montgomery
n the "Montgomery ladder".

```
rmult(n,x1):
3,z3 = 1,0,x1,1
n reversed(range(maxnbits)):
 1 & (n >> i)
 = cswap(x2,x3,bit)
 = cswap(z2,z3,bit)
 = ((x2*x3-z2*z3)^2,
   x1*(x2*z3-z2*x3)^2)
 = ((x2^2-z2^2)^2,
   4*x2*z2*(x2^2+A*x2*z2+z2^2))
 = cswap(x2,x3,bit)
 = cswap(z2,z3,bit)
x2*z2^(p-2)
```

Curve selection

How to defend yourself against
an attacker armed with a mathematician:

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

You can pi

What your
No known
ECC user's
("Elliptic-c

Example of
Standard b
has huge p
i.e., exactly

All criteria
See our eva
safecurve

strass: Montgomery
gomery ladder".

```
:
x1,1
range(maxnbits)):
i)
,x3,bit)
,z3,bit)
z2*z3)^2,
3-z2*x3)^2)
2^2)^2,
(x2^2+A*x2*z2+z2^2))
,x3,bit)
,z3,bit)
```

Curve selection

How to defend yourself against
an attacker armed with a mathematician:

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

You can pick any of the

What your chosen stan
No known attack will c
ECC user's secret key f
("Elliptic-curve discrete

Example of criterion in
Standard base point (x
has huge prime "order"
i.e., exactly ℓ different

All criteria are compute
See our evaluation site
safecurves.cr.yp.to

gomery
er".

ts)):

z2+z2^2))

Curve selection

How to defend yourself against an attacker armed with a mathematician:

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

You can pick any of these standards

What your chosen standard achieves
No known attack will compute
ECC user's secret key from public k
("Elliptic-curve discrete-log problem

Example of criterion in all standards
Standard base point (x, y)
has huge prime "order" ℓ,
i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
safecurves.cr.yp.to

Curve selection

How to defend yourself against
an attacker armed with a mathematician:

1999 ANSI X9.62.
2000 IEEE P1363.
2000 Certicom SEC 2.
2000 NIST FIPS 186-2.
2001 ANSI X9.63.
2005 Brainpool.
2005 NSA Suite B.
2010 Certicom SEC 2 v2.
2010 OSCCA SM2.
2011 ANSSI FRP256V1.

You can pick any of these standards.

What your chosen standard achieves:
No known attack will compute
ECC user's secret key from public key.
("Elliptic-curve discrete-log problem.")

Example of criterion in all standards:
Standard base point (x, y)
has huge prime "order" ℓ,
i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
`safecurves.cr.yp.to`

ction

fend yourself against
r armed with a mathematician:

X9.62.
P1363.
com SEC 2.
FIPS 186-2.
X9.63.
pool.
Suite B.
com SEC 2 v2.
CA SM2.
SI FRP256V1.

You can pick any of these standards.

What your chosen standard achieves:
No known attack will compute
ECC user's secret key from public key.
("Elliptic-curve discrete-log problem.")

Example of criterion in all standards:
Standard base point (x, y)
has huge prime "order" ℓ,
i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
`safecurves.cr.yp.to`

You do eve

You pick th
`brainpool`
$y^2 = x^3 -$
standard ba

This curve
with Edwar
So you che
in the Weie

You make
It's horrend
but it's sec

against
a mathematician:

.

2.

1.

You can pick any of these standards.

What your chosen standard achieves:
No known attack will compute
ECC user's secret key from public key.
("Elliptic-curve discrete-log problem.")

Example of criterion in all standards:
Standard base point (x, y)
has huge prime "order" ℓ,
i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
safecurves.cr.yp.to

You do everything right

You pick the Brainpool
brainpoolP256t1: hu
$y^2 = x^3 - 3x +$ somehu
standard base point.

This curve isn't compat
with Edwards or Montg
So you check and test e
in the Weierstrass form

You make it all constan
It's horrendously slow,
but it's secure.

tician:

You can pick any of these standards.

What your chosen standard achieves:
No known attack will compute ECC user's secret key from public key.
("Elliptic-curve discrete-log problem.")

Example of criterion in all standards:
Standard base point (x, y) has huge prime "order" ℓ, i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
`safecurves.cr.yp.to`

You do everything right.

You pick the Brainpool curve `brainpoolP256t1`: huge prime p, $y^2 = x^3 - 3x + \text{somehugenumber}$, standard base point.

This curve isn't compatible with Edwards or Montgomery.
So you check and test every case in the Weierstrass formulas.

You make it all constant-time.
It's horrendously slow, but it's secure.

You can pick any of these standards.

What your chosen standard achieves:
No known attack will compute
ECC user's secret key from public key.
("Elliptic-curve discrete-log problem.")

Example of criterion in all standards:
Standard base point (x, y)
has huge prime "order" ℓ,
i.e., exactly ℓ different multiples.

All criteria are computer-verifiable.
See our evaluation site for scripts:
safecurves.cr.yp.to

You do everything right.

You pick the Brainpool curve
`brainpoolP256t1`: huge prime p,
$y^2 = x^3 - 3x +$ somehugenumber,
standard base point.

This curve isn't compatible
with Edwards or Montgomery.
So you check and test every case
in the Weierstrass formulas.

You make it all constant-time.
It's horrendously slow,
but it's secure.

ck any of these standards.

chosen standard achieves:
attack will compute
secret key from public key.
urve discrete-log problem.")

f criterion in all standards:
base point (x, y)
rime "order" ℓ,
y ℓ different multiples.

are computer-verifiable.
aluation site for scripts:
es.cr.yp.to

You do everything right.

You pick the Brainpool curve `brainpoolP256t1`: huge prime p, $y^2 = x^3 - 3x + \text{somehugenumber}$, standard base point.

This curve isn't compatible with Edwards or Montgomery. So you check and test every case in the Weierstrass formulas.

You make it all constant-time. It's horrendously slow, but it's secure.

Actually, it

The attack

$x' =$ 1025b3
1e86be

$y' =$ 12ace5
d123d5

You compu
using the W
You encryp
with a hash

ese standards.

dard achieves:

ompute

rom public key.

e-log problem.")

all standards:

, y)

ℓ,

multiples.

er-verifiable.

for scripts:

o

You do everything right.

You pick the Brainpool curve `brainpoolP256t1`: huge prime p, $y^2 = x^3 - 3x +$ somehugenumber, standard base point.

This curve isn't compatible with Edwards or Montgomery. So you check and test every case in the Weierstrass formulas.

You make it all constant-time. It's horrendously slow, but it's secure.

Actually, it's not. **You'**

The attacker sent you (

$x' =$ `1025b35abab9150d8677`
`1e86bec6c6bac120535e`

$y' =$ `12ace5eeae9a5b0bca8e`
`d123d55f68100099b65a`

You computed "shared

using the Weierstrass fo

You encrypted data usi

with a hash of $a(x', y')$

s.

s:

ey.

.")

s:

You do everything right.

You pick the Brainpool curve `brainpoolP256t1`: huge prime p, $y^2 = x^3 - 3x + \text{somehugenumber}$, standard base point.

This curve isn't compatible with Edwards or Montgomery. So you check and test every case in the Weierstrass formulas.

You make it all constant-time. It's horrendously slow, but it's secure.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` a

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x'$ using the Weierstrass formulas. You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

You do everything right.

You pick the Brainpool curve `brainpoolP256t1`: huge prime p, $y^2 = x^3 - 3x + \text{somehugenumber}$, standard base point.

This curve isn't compatible with Edwards or Montgomery. So you check and test every case in the Weierstrass formulas.

You make it all constant-time. It's horrendously slow, but it's secure.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas. You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

You do everything right.

You pick the Brainpool curve `brainpoolP256t1`: huge prime p, $y^2 = x^3 - 3x + \text{somehugenumber}$, standard base point.

This curve isn't compatible with Edwards or Montgomery. So you check and test every case in the Weierstrass formulas.

You make it all constant-time. It's horrendously slow, but it's secure.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' = $ `1025b35abab9150d86770f6bda12f8ec1e86bec6c6bac120535e4134fea87831` and

$y' = $ `12ace5eeae9a5b0bca8ed1c0f9540d05d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas. You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

What you never noticed: (x', y') isn't his public key $b(x, y)$; it isn't even a point on `brainpoolP256t1`; it's a point on $y^2 = x^3 - 3x + 5$ of order only 4999.

erything right.

he Brainpool curve
lP256t1: huge prime p,
$3x+$ somehugenumber,
ase point.

isn't compatible
rds or Montgomery.
eck and test every case
erstrass formulas.

it all constant-time.
dously slow,
cure.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$
using the Weierstrass formulas.
You encrypted data using AES-GCM
with a hash of $a(x', y')$ as a key.

What you never noticed:
(x', y') isn't his public key $b(x, y)$;
it isn't even a point on `brainpoolP256t1`;
it's a point on $y^2 = x^3 - 3x + 5$
of order only 4999.

Your formu
because th

Addition on
$y^2 = x^3 + a_4$
for $x_1 \neq x_2$, (
(x_3, y_3) with
$y_3 = \lambda(x_1 -$
$\lambda = (y_2 - y_1$
for $y_1 \neq 0$, (
(x_3, y_3) with
$y_3 = \lambda(x_1 -$
$\lambda = (3x_1^2 + a$
$(x_1, y_1) + (x_1$
$(x_1, y_1) + \infty$
$\infty + (x_2, y_2)$
$\infty + \infty = \infty$
Messy to imp

t.

curve

ge prime p,

ugenumber,

tible

gomery.

every case

ulas.

nt-time.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas.
You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

What you never noticed:
(x', y') isn't his public key $b(x, y)$;
it isn't even a point on `brainpoolP256t1`;
it's a point on $y^2 = x^3 - 3x + 5$
of order only 4999.

Your formulas worked f
because they work for a

Addition on Weierstrass curves
$y^2 = x^3 + a_4 x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2)$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x$
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x$
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and
$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas.
You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

What you never noticed:
(x', y') isn't his public key $b(x, y)$;
it isn't even a point on `brainpoolP256t1`;
it's a point on $y^2 = x^3 - 3x + 5$
of order only 4999.

Your formulas worked for $y^2 = x^3 -$
because they work for any $y^2 = x^3$

Addition on Weierstrass curves
$y^2 = x^3 + a_4x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6

Actually, it's not. **You're screwed.**

The attacker sent you (x', y') with

$x' =$ `1025b35abab9150d86770f6bda12f8ec` `1e86bec6c6bac120535e4134fea87831` and

$y' =$ `12ace5eeae9a5b0bca8ed1c0f9540d05` `d123d55f68100099b65a99ac358e3a75`.

You computed "shared secret" $a(x', y')$ using the Weierstrass formulas.
You encrypted data using AES-GCM with a hash of $a(x', y')$ as a key.

What you never noticed:
(x', y') isn't his public key $b(x, y)$;
it isn't even a point on `brainpoolP256t1`;
it's a point on $y^2 = x^3 - 3x + 5$
of order only 4999.

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves
$y^2 = x^3 + a_4 x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6 here!

’s not. **You’re screwed.**

ker sent you (x', y') with

`5abab9150d86770f6bda12f8ec`
`c6c6bac120535e4134fea87831` and
`eeae9a5b0bca8ed1c0f9540d05`
`5f68100099b65a99ac358e3a75`.

uted “shared secret” $a(x', y')$
Weierstrass formulas.
pted data using AES-GCM
h of $a(x', y')$ as a key.

never noticed:
’t his public key $b(x, y)$;
n a point on `brainpoolP256t1`;
t on $y^2 = x^3 - 3x + 5$
nly 4999.

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves
$y^2 = x^3 + a_4 x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6 here!

Why this m
$a(x', y')$ is
The attack
compares t
learns your

re screwed.

(x', y') with

`0f6bda12f8ec`
`4134fea87831` and
`d1c0f9540d05`
`99ac358e3a75`.

secret" $a(x', y')$

ormulas.

ng AES-GCM

as a key.

d:

key $b(x, y)$;

`brainpoolP256t1`;

$- 3x + 5$

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves
$y^2 = x^3 + a_4 x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6 here!

Why this matters: $(x',$
$a(x', y')$ is determined
The attacker tries all 4
compares to the AES-G
learns your secret a mo

nd

$, y')$

M

`P256t1;`

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves
$y^2 = x^3 + a_4x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6 here!

Why this matters: (x', y') has order
$a(x', y')$ is determined by a mod 49
The attacker tries all 4999 possibilit
compares to the AES-GCM output,
learns your secret a mod 4999.

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves
$y^2 = x^3 + a_4x + a_6$:
for $x_1 \neq x_2$, $(x_1, y_1) + (x_2, y_2) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (y_2 - y_1)/(x_2 - x_1)$;
for $y_1 \neq 0$, $(x_1, y_1) + (x_1, y_1) =$
(x_3, y_3) with $x_3 = \lambda^2 - x_1 - x_2$,
$y_3 = \lambda(x_1 - x_3) - y_1$,
$\lambda = (3x_1^2 + a_4)/2y_1$;
$(x_1, y_1) + (x_1, -y_1) = \infty$;
$(x_1, y_1) + \infty = (x_1, y_1)$;
$\infty + (x_2, y_2) = (x_2, y_2)$;
$\infty + \infty = \infty$.
Messy to implement and test.

No a_6 here!

Why this matters: (x', y') has order 4999.
$a(x', y')$ is determined by a mod 4999.
The attacker tries all 4999 possibilities, compares to the AES-GCM output, learns your secret a mod 4999.

Your formulas worked for $y^2 = x^3 - 3x + 5$ because they work for any $y^2 = x^3 - 3x + a_6$:

Addition on Weierstrass curves $y^2 = x^3 + a_4x + a_6$:

$$\left.\begin{array}{l}
\text{for } x_1 \neq x_2,\ (x_1, y_1) + (x_2, y_2) = \\
(x_3, y_3) \text{ with } x_3 = \lambda^2 - x_1 - x_2, \\
y_3 = \lambda(x_1 - x_3) - y_1, \\
\lambda = (y_2 - y_1)/(x_2 - x_1); \\
\text{for } y_1 \neq 0,\ (x_1, y_1) + (x_1, y_1) = \\
(x_3, y_3) \text{ with } x_3 = \lambda^2 - x_1 - x_2, \\
y_3 = \lambda(x_1 - x_3) - y_1, \\
\lambda = (3x_1^2 + a_4)/2y_1; \\
(x_1, y_1) + (x_1, -y_1) = \infty; \\
(x_1, y_1) + \infty = (x_1, y_1); \\
\infty + (x_2, y_2) = (x_2, y_2); \\
\infty + \infty = \infty. \\
\text{Messy to implement and test.}
\end{array}\right\} \text{No } a_6 \text{ here!}$$

Why this matters: (x', y') has order 4999. $a(x', y')$ is determined by a mod 4999. The attacker tries all 4999 possibilities, compares to the AES-GCM output, learns your secret a mod 4999.

Attacker then tries again with

$x' =$ `9bc001a0d2d5c43863aadb0f881df3bbaf3a5ea81eedd2385e6525521aa8b1e2` and

$y' =$ `0d124e9e94dcede52aa0e3bcac1852cfed28eb86039c0d8e0cfaa4ae703eac07`,

a point of order 19559 on $y^2 = x^3 - 3x + 211$; learns your secret a mod 19559.

Etc. Uses “Chinese remainder theorem” to combine this information.

ulas worked for $y^2 = x^3 - 3x + 5$

ey work for any $y^2 = x^3 - 3x + a_6$:

Weierstrass curves

$_4x + a_6$:

$(x_1, y_1) + (x_2, y_2) =$

$x_3 = \lambda^2 - x_1 - x_2,$

$x_3) - y_1,$

$)/(x_2 - x_1);$

$x_1, y_1) + (x_1, y_1) =$

$x_3 = \lambda^2 - x_1 - x_2,$

$x_3) - y_1,$

$a_4)/2y_1;$

$_1, -y_1) = \infty;$

$= (x_1, y_1);$

$= (x_2, y_2);$

∞.

plement and test.

No a_6 here!

Why this matters: (x', y') has order 4999.

$a(x', y')$ is determined by a mod 4999.

The attacker tries all 4999 possibilities,

compares to the AES-GCM output,

learns your secret a mod 4999.

Attacker then tries again with

$x' =$ `9bc001a0d2d5c43863aadb0f881df3bb` `af3a5ea81eedd2385e6525521aa8b1e2` and

$y' =$ `0d124e9e94dcede52aa0e3bcac1852cf` `ed28eb86039c0d8e0cfaa4ae703eac07`,

a point of order 19559

on $y^2 = x^3 - 3x + 211$;

learns your secret a mod 19559.

Etc. Uses "Chinese remainder theorem"

to combine this information.

Traditional

Blame the

"You shoul

the incomi

and had th

(And mayb

or $y^2 = x^3 - 3x + 5$

any $y^2 = x^3 - 3x + a_6$:

s

=

2,

=

2,

No a_6 here!

Why this matters: (x', y') has order 4999.
$a(x', y')$ is determined by a mod 4999.
The attacker tries all 4999 possibilities,
compares to the AES-GCM output,
learns your secret a mod 4999.

Attacker then tries again with

$x' =$ `9bc001a0d2d5c43863aadb0f881df3bb` `af3a5ea81eedd2385e6525521aa8b1e2` and

$y' =$ `0d124e9e94dcede52aa0e3bcac1852cf` `ed28eb86039c0d8e0cfaa4ae703eac07`,

a point of order 19559
on $y^2 = x^3 - 3x + 211$;
learns your secret a mod 19559.

Etc. Uses "Chinese remainder theorem"
to combine this information.

Traditional response to
Blame the implementor

"You should have check
the incoming (x', y') wa
and had the right order
(And maybe paid paten

$- 3x + 5$

$-3x + a_6$:

here!

Why this matters: (x', y') has order 4999.
$a(x', y')$ is determined by a mod 4999.
The attacker tries all 4999 possibilities,
compares to the AES-GCM output,
learns your secret a mod 4999.

Attacker then tries again with

$x' =$ `9bc001a0d2d5c43863aadb0f881df3bb` `af3a5ea81eedd2385e6525521aa8b1e2` and

$y' =$ `0d124e9e94dcede52aa0e3bcac1852cf` `ed28eb86039c0d8e0cfaa4ae703eac07`,

a point of order 19559
on $y^2 = x^3 - 3x + 211$;
learns your secret a mod 19559.

Etc. Uses "Chinese remainder theorem"
to combine this information.

Traditional response to this security
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the rig
and had the right order."
(And maybe paid patent fees to Cer

Why this matters: (x', y') has order 4999.
$a(x', y')$ is determined by a mod 4999.
The attacker tries all 4999 possibilities,
compares to the AES-GCM output,
learns your secret a mod 4999.

Attacker then tries again with

$x' =$ `9bc001a0d2d5c43863aadb0f881df3bb`
`af3a5ea81eedd2385e6525521aa8b1e2` and

$y' =$ `0d124e9e94dcede52aa0e3bcac1852cf`
`ed28eb86039c0d8e0cfaa4ae703eac07`,

a point of order 19559
on $y^2 = x^3 - 3x + 211$;
learns your secret a mod 19559.

Etc. Uses "Chinese remainder theorem"
to combine this information.

Traditional response to this security failure:
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the right curve
and had the right order."
(And maybe paid patent fees to Certicom.)

Why this matters: (x', y') has order 4999.
$a(x', y')$ is determined by a mod 4999.
The attacker tries all 4999 possibilities,
compares to the AES-GCM output,
learns your secret a mod 4999.

Attacker then tries again with

$x' =$ `9bc001a0d2d5c43863aadb0f881df3bb` `af3a5ea81eedd2385e6525521aa8b1e2` and

$y' =$ `0d124e9e94dcede52aa0e3bcac1852cf` `ed28eb86039c0d8e0cfaa4ae703eac07`,

a point of order 19559
on $y^2 = x^3 - 3x + 211$;
learns your secret a mod 19559.

Etc. Uses "Chinese remainder theorem"
to combine this information.

Traditional response to this security failure:
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the right curve
and had the right order."
(And maybe paid patent fees to Certicom.)

But it's much better to
design the system without traps.

Never send uncompressed (x, y).
Design protocols to compress
one coordinate down to 1 bit, or 0 bits!
Drastically limits possibilities
for attacker to choose points.

matters: (x', y') has order 4999.
determined by a mod 4999.
ker tries all 4999 possibilities,
to the AES-GCM output,
secret a mod 4999.

hen tries again with
a0d2d5c43863aadb0f881df3bb
a81eedd2385e6525521aa8b1e2 and
9e94dcede52aa0e3bcac1852cf
86039c0d8e0cfaa4ae703eac07'
order 19559
$^3 - 3x + 211$;
secret a mod 19559.

"Chinese remainder theorem"
e this information.

Traditional response to this security failure:
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the right curve
and had the right order."
(And maybe paid patent fees to Certicom.)

But it's much better to
design the system without traps.

Never send uncompressed (x, y).
Design protocols to compress
one coordinate down to 1 bit, or 0 bits!
Drastically limits possibilities
for attacker to choose points.

Always mu

If the curve
and the ba
then c is ca
and $c \cdot \ell$ is

Design DH

Always ch

Montgome
but modify
curve order
to be large

DH protoc
are robust
every comr

y') has order 4999.
by a mod 4999.
999 possibilities,
GCM output,
d 4999.

in with
db0f881df3bb
25521aa8b1e2 and
e3bcac1852cf
a4ae703eac07'

;
d 19559.

nainder theorem"
ation.

Traditional response to this security failure:
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the right curve
and had the right order."
(And maybe paid patent fees to Certicom.)

But it's much better to
design the system without traps.

Never send uncompressed (x, y).
Design protocols to compress
one coordinate down to 1 bit, or 0 bits!
Drastically limits possibilities
for attacker to choose points.

Always multiply DH s

If the curve has $c \cdot \ell$ po
and the base point P h
then c is called the cofa
and $c \cdot \ell$ is called the cu

Design DH protocols to

Always choose twist-s

Montgomery formulas u
but modifying B gives
curve orders. Require b
to be large primes time

DH protocols with all o
are robust against
every common DH imp

r 4999.
99.
ties,

nd

rem"

Traditional response to this security failure: Blame the implementor.

"You should have checked that the incoming (x', y') was on the right curve and had the right order." (And maybe paid patent fees to Certicom.)

But it's much better to *design the system without traps*.

Never send uncompressed (x, y)**.** Design protocols to compress one coordinate down to 1 bit, or 0 bits! Drastically limits possibilities for attacker to choose points.

Always multiply DH scalar by cof

If the curve has $c \cdot \ell$ points and the base point P has order ℓ then c is called the cofactor and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by

Always choose twist-secure curve

Montgomery formulas use only A, but modifying B gives only *two* diff curve orders. Require both of these to be large primes times small cofac

DH protocols with all of these prote are robust against every common DH implementation

Traditional response to this security failure:
Blame the implementor.

"You should have checked that
the incoming (x', y') was on the right curve
and had the right order."
(And maybe paid patent fees to Certicom.)

But it's much better to
design the system without traps.

Never send uncompressed (x, y).
Design protocols to compress
one coordinate down to 1 bit, or 0 bits!
Drastically limits possibilities
for attacker to choose points.

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points
and the base point P has order ℓ
then c is called the cofactor
and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A,
but modifying B gives only *two* different
curve orders. Require both of these orders
to be large primes times small cofactors.

DH protocols with all of these protections
are robust against
every common DH implementation error.

response to this security failure:
implementor.

ld have checked that
ng (x', y') was on the right curve
e right order."
e paid patent fees to Certicom.)

uch better to
system without traps.

d uncompressed (x, y).
tocols to compress
nate down to 1 bit, or 0 bits!
limits possibilities
r to choose points.

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points
and the base point P has order ℓ
then c is called the cofactor
and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A,
but modifying B gives only *two* different
curve orders. Require both of these orders
to be large primes times small cofactors.

DH protocols with all of these protections
are robust against
every common DH implementation error.

<u>ECC stand</u>

Fix the sta
so that **sim**
are **secure**

Bonus: nex
Curve2551

2010.03 Ac
"Curve255
appear on
[Google] w

this security failure:
.

ked that
as on the right curve
."
nt fees to Certicom.)

o
out traps.

essed (x, y).
mpress
o 1 bit, or 0 bits!
bilities
points.

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points
and the base point P has order ℓ
then c is called the cofactor
and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A,
but modifying B gives only *two* different
curve orders. Require both of these orders
to be large primes times small cofactors.

DH protocols with all of these protections
are robust against
every common DH implementation error.

ECC standards: the nex

Fix the standard curves
so that **simple** impleme
are **secure** implementat

Bonus: next-generation
Curve25519 are faster t

2010.03 Adam Langley,
"Curve25519 doesn't cu
appear on IANA's list .
[Google] would like to s

failure:

ht curve

rticom.)

bits!

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points
and the base point P has order ℓ
then c is called the cofactor
and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A,
but modifying B gives only *two* different
curve orders. Require both of these orders
to be large primes times small cofactors.

DH protocols with all of these protections
are robust against
every common DH implementation error.

ECC standards: the next generation

Fix the standard curves and protoco
so that **simple** implementations
are **secure** implementations.

Bonus: next-generation curves such
Curve25519 are faster than the stan

2010.03 Adam Langley, TLS mailin
"Curve25519 doesn't currently
appear on IANA's list . . . and we
[Google] would like to see it include

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points and the base point P has order ℓ then c is called the cofactor and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A, but modifying B gives only *two* different curve orders. Require both of these orders to be large primes times small cofactors.

DH protocols with all of these protections are robust against every common DH implementation error.

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

Always multiply DH scalar by cofactor.

If the curve has $c \cdot \ell$ points
and the base point P has order ℓ
then c is called the cofactor
and $c \cdot \ell$ is called the curve order.

Design DH protocols to multiply by c.

Always choose twist-secure curves.

Montgomery formulas use only A,
but modifying B gives only *two* different
curve orders. Require both of these orders
to be large primes times small cofactors.

DH protocols with all of these protections
are robust against
every common DH implementation error.

ECC standards: the next generation

Fix the standard curves and protocols
so that **simple** implementations
are **secure** implementations.

Bonus: next-generation curves such as
Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list:
"Curve25519 doesn't currently
appear on IANA's list . . . and we
[Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange
specify a procedure to generate a
next-generation curve at any security level.

ultiply DH scalar by cofactor.**

e has $c \cdot \ell$ points
se point P has order ℓ
alled the cofactor
called the curve order.

protocols to multiply by c.

oose twist-secure curves.

ry formulas use only A,
ving B gives only *two* different
rs. Require both of these orders
primes times small cofactors.

ols with all of these protections
against
mon DH implementation error.

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange specify a procedure to generate a next-generation curve at any security level.

2013.09 Pa
that's rece
curves, is i
adding cur

scalar by cofactor.**

oints
as order ℓ
actor
urve order.

o multiply by c.

secure curves.

use only A,
only *two* different
both of these orders
s small cofactors.

of these protections

lementation error.

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange specify a procedure to generate a next-generation curve at any security level.

2013.09 Patrick Pelletie
that's recently been cas
curves, is it time to rev
adding curve25519 as a

factor.

c.

es.

ferent
orders
ctors.

ections

error.

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange specify a procedure to generate a next-generation curve at any security level.

2013.09 Patrick Pelletier: "Given th
that's recently been cast on the NIS
curves, is it time to revive the idea
adding curve25519 as a named curv

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange specify a procedure to generate a next-generation curve at any security level.

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

ECC standards: the next generation

Fix the standard curves and protocols so that **simple** implementations are **secure** implementations.

Bonus: next-generation curves such as Curve25519 are faster than the standards!

2010.03 Adam Langley, TLS mailing list: "Curve25519 doesn't currently appear on IANA's list . . . and we [Google] would like to see it included."

2013.05 Bernstein–Krasnova–Lange specify a procedure to generate a next-generation curve at any security level.

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

ards: the next generation

ndard curves and protocols
nple implementations
implementations.

xt-generation curves such as
9 are faster than the standards!

dam Langley, TLS mailing list:
19 doesn't currently
IANA's list . . . and we
ould like to see it included."

ernstein–Krasnova–Lange
rocedure to generate a
ation curve at any security level.

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

2013.09 Ni
"Agreed, w
because of
not due to
ECDH curv

xt generation

and protocols
entations
tions.

curves such as
than the standards!

TLS mailing list:
urrently
. . and we
see it included."

snova–Lange
generate a
t any security level.

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

2013.09 Nico Williams:
"Agreed, we need curve
because of its technical
not due to any FUD ab
ECDH curves that we h

n

ols

as
ndards!

g list:

d.”

ty level.

2013.09 Patrick Pelletier: “Given the doubt that’s recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?”

2013.09 Douglas Stebila: Reasons to support Curve25519 are “efficiency and resistance to side-channel attacks” rather than concerns about backdoors.

2013.09 Nick Mathewson: “In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues.”

2013.09 Nico Williams: “Agreed, we need curve25519 ciphe because of its technical advantages, not due to any FUD about the othe ECDH curves that we have.”

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 Patrick Pelletier: “Given the doubt that’s recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?”

2013.09 Douglas Stebila: Reasons to support Curve25519 are “efficiency and resistance to side-channel attacks” rather than concerns about backdoors.

2013.09 Nick Mathewson: “In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues.”

2013.09 Nico Williams:
“Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have.”

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.09 Patrick Pelletier: "Given the doubt that's recently been cast on the NIST curves, is it time to revive the idea of adding curve25519 as a named curve?"

2013.09 Douglas Stebila: Reasons to support Curve25519 are "efficiency and resistance to side-channel attacks" rather than concerns about backdoors.

2013.09 Nick Mathewson: "In the FOSS cryptography world nowadays, I see many more new users of curve25519 than of the NIST curves, because of efficiency and ease-of-implementation issues."

2013.09 Nico Williams: "Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

atrick Pelletier: "Given the doubt
ntly been cast on the NIST
t time to revive the idea of
ve25519 as a named curve?"

ouglas Stebila: Reasons to
urve25519 are "efficiency
nce to side-channel attacks"
n concerns about backdoors.

ck Mathewson: "In the
tography world nowadays, I see
e new users of curve25519 than
T curves, because of efficiency
f-implementation issues."

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 W

er: "Given the doubt
st on the NIST
ive the idea of
n named curve?"

a: Reasons to
e "efficiency
channel attacks"
bout backdoors.

on: "In the
rld nowadays, I see
of curve25519 than
cause of efficiency
ation issues."

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce S

ne doubt
ST
of
ve?”

o

ks”
ors.

s, I see
9 than
ciency

2013.09 Nico Williams:
“Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have.”

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves s

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 "chosen curves", including 13 next-generation curves.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 "chosen curves", including 13 next-generation curves.

2014.06 CFRG announces change of leadership.

2013.09 Nico Williams:
"Agreed, we need curve25519 cipher suites because of its technical advantages, not due to any FUD about the other ECDH curves that we have."

2013.09 Simon Josefsson writes an Internet-Draft. Active discussion on TLS mailing list.

2013.09 We announce next-generation Curve41417, computed for Silent Circle.

2013.10 Aranha–Barreto–Pereira–Ricardini announce next-generation curves computed at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 "chosen curves", including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA "will work with the two new chairs until he retires next year".

co Williams:

ve need curve25519 cipher suites

its technical advantages,

any FUD about the other

ves that we have."

mon Josefsson writes an Internet-

ive discussion on TLS mailing list.

e announce next-generation

7, computed for Silent Circle.

ranha–Barreto–Pereira–Ricardini

next-generation curves

at various security levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 "chosen curves", including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA "will work with the two new chairs until he retires next year".

[. . . more t

e25519 cipher suites
advantages,
bout the other
have."

on writes an Internet-
n on TLS mailing list.

next-generation
for Silent Circle.

o–Pereira–Ricardini
on curves
curity levels.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 "chosen curves", including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA "will work with the two new chairs until he retires next year".

[. . . more than 1000 em

r suites

er

Internet-
ailing list.

”on
rcle.

cardini

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 “chosen curves”, including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA “will work with the two new chairs until he retires next year”.

[. . . more than 1000 email messages

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 “chosen curves”, including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA “will work with the two new chairs until he retires next year”.

[. . . more than 1000 email messages . . .]

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 “chosen curves”, including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA “will work with the two new chairs until he retires next year”.

[. . . more than 1000 email messages . . .]

2014.12 CFRG discussion is continuing.

2013.10 We announce SafeCurves site.

2013.11 Aranha–Barreto–Pereira–Ricardini announce next-generation E-521.

2014.01 Discussion spreads to IRTF CFRG.

2014.01 Mike Hamburg announces next-generation Ed448-Goldilocks.

2014.02 Microsoft announces 26 “chosen curves”, including 13 next-generation curves.

2014.06 CFRG announces change of leadership. Previous co-chair from NSA “will work with the two new chairs until he retires next year”.

[. . . more than 1000 email messages . . .]

2014.12 CFRG discussion is continuing.

Sage scripts to verify criteria for ECDLP security and ECC security: `safecurves.cr.yp.to`

Analysis of manipulability of various curve-generation methods: `safecurves.cr.yp.to/bada55.html`

Many computer-verified addition formulas: `hyperelliptic.org/EFD/`

Python scripts for this talk: `ecchacks.cr.yp.to`